通信情報保護學會論文誌
第 8 卷 第 1 號, 1998. 3

시스톨릭 어레이상에서 고속 모듈러 지수 연산

이 건 직*, 허 영 준* 유 기 영*

Fast Modular Exponentiation on a Systolic Array

Keon-Jik Lee, Young-Jun Heo, Kee-Young Yoo

요 약

본 논문에서는 모듈러 지수승시에 요구되는 모듈러 곱셈의 반복 횟수를 줄이기 위해 *SE(m)*기법을 제안하며 지수를 *SE(m)* 표현과 시스톨릭 *SE(m)* 표현으로 변환한다. 그리고 변환된 시스톨릭 SE(*m*) 표현으로부터 모듈러 지수연산을 위한 선형 시스톨릭 어레이를 제시한다. 제안된 기법은 기존의 방법보다 소프트웨어로 구현시에 선 계산시에 필요한 기억 장소의 크기를 줄였으며, 선형 시스톨릭 어레이로 구현시에 기존의 방법들보다 처리기의 개수를 감소시키며, 처리기내에 필요한 기억 장소의 크기를 줄였다. 수정된 부호화 디지트 기법과 비교하면 처리기의 개수를 24%정도 줄일 수 있다.

Abstract

To reduce the repetitive number of the modular multiplication required by the modular exponentiation, this paper introduces *SE(m)* method and transforms the exponent into *SE(m)* representation and systolic *SE(m)* representation. And the linear systolic array for the modular exponentiation is presented from the systolic *SE(m)* representation. It shows that the proposed method requires smaller memory size for precomputation in implementing software, and reduces the number of the processing elements in implementing the linear systolic array. Moreover, it reduces the memory size in the processing element. Compared to the modified signed digit algorithm, the proposed method reduces the number of the processing elements by 24%.

***Keywords**: modular exponentiation, RSA, systolic array.*

---

* 경북대학교 컴퓨터 공학과

Ⅰ. 서 론

정보화 사회에서 급격히 증가하고 있는 전자 은행 거래(electronic banking transaction)나 전자우편과 같은 새로운 형태의 데이터 통신에서 안전성(security)의 문제를 해결하기 위해 암호화 알고리즘과 그것의 효율적인 구현에 대한 연구가 이루어지고 있다. 여러 암호화 방법 중에서 1976년 W. Diffie와 M. E. Hellman은 공개키 암호 시스템의 개념을 처음으로 제안하였다[1]. 그후 안전성을 인정받고 있는 공개키 암호 시스템은 1978년 Rivest, Shamir, Adleman이 제안한 RSA암호 시스템이다[2]. RSA암호화 시스템에서는 실제 데이터 메시지 M을 전송하지 않고 송신자는 수신자의 공개 키(public key) e를 사용하여 암호화된 메시지 $C = M^e (mod\ N)$을 보낸다. 수신자 측에서는 자신만의 비밀 키(private key) d를 이용해서 원본 데이터 메시지 $M = C^d\ (mod\ N)$을 복원한다.

공개키 암호 시스템의 암호화와 복호화 과정은 아주 큰 수에 의한 모듈러 지수연산(modular exponentiation) $M^e (mod\ N)$로 표현되며, 이 연산은 $A \cdot B\ mod\ N$형태의 모듈러 곱셈(modular multiplication)을 반복적으로 수행함으로써 계산된다. 그러나 암호화와 복호화시에 512비트 이상의 아주 큰 수의 모듈러 지수연산을 수행하여야 하므로 처리속도의 지연이 큰 문제가 된다. 처리 속도를 높이기 위해서는 내재된 반복적인 모듈러 곱셈의 수행 횟수를 줄이거나 모듈러 곱셈 연산 자체의 처리시간을 줄이는 크게 두 가지의 방법이 제시되고 있다[3, 4].

모듈러 지수승에 내재된 모듈러 곱셈의 횟수를 줄이기 위한 지수 e의 다양한 표현법이 제안되었다. 수정된 부호화 디지트(modified signed digit)표현은 메시지 m^{-1}을 미리 계산하고 $n/3$개의 모듈러 곱셈을 필요로 한다[5, 6]. 열 치환(string replacement) 표현에 기반한 k-SR 알고리즘은 모듈러 곱셈의 수를 최대 $n/4$로 줄이는 것이 가능하다[7, 8]. 최근에 더 나은 효율을 가진 $SS(l)$ 알고리즘이 Lam과 Hui에 의해 제안되었다[9, 10]. $SS(l)$ 알고리즘은 미리 계산한 값들을 저장할 공간이 충분하다면 모듈러 곱셈의 수를 $n/(l+1)$로 줄일 수 있다. 여기서 l은 미리 계산되어 지는 지수의 최대 길이이다.

본 논문에서는 $SS(l)$을 변형한 $SE(m)$ 기법을 이용하여 이진 지수표현을 모듈러 곱셈 횟수를 최소화하고 병렬 처리가 가능한 수 표현으로 변환시킨다. $SE(m)$방법은 e의 이진 표현에서 고정된 m개의 크기내에서 불연속적인 홀수로 표현 가능한 비트 패턴을 찾아서 대치하는 것이다. 변환 후에 메시지의 선 계산시에 필요로 하는 홀수 지수승을 알 수 있기 때문에 2^m보다 작은 모든 홀수 지수승을 계산할 필요가 없다. 결과적으로 선 처리한 메시지의 지수승의 값들을 저장하는 기억공간의 크기를 줄일 수 있다. 그리고, 이를 시스톨릭 어레이로 구현하기 위해 시스톨릭 $SE(m)$ 표현으로 변환시켜 모듈러 지수연산을 수행하는 선형 시스톨릭 어레이(linear systolic array)를 제안한다. 이때 선 계산한 값들을 처리기내의 메모리에 보관하지 않고 다른 입력값들과 같이 이웃하는 처리기로 흐르도록 한다. 한편 본 논문에서 사용하는 모듈러 곱셈기는 몽고메리 알고리즘을 이용하여 김[11]이 제안한 선형 시스톨릭 모듈러 곱셈기(linear systolic modular multiplier)를 사용하기로 한다.

본 논문의 구성은 다음과 같다. Ⅱ장에서 기존의 모듈러 지수승 알고리즘에 대해 살펴보고, $SE(m)$ 모듈러 지수승 알고리즘을 제안한다. Ⅲ장에서 $SE(m)$ 알고리즘을 시스톨릭 어레이에 적용 시키며, Ⅳ장에서는 제안된 방법을 기존의 알고리즘과 비교 분석하며, Ⅴ장에서 결론을 내린다.

Ⅱ. 모듈러 지수승 알고리즘

아주 큰 수의 모듈러 지수연산은 RSA나 ElGamal과 같은 공개키 암호화 시스템의 기본적인 연산이며 일반 사용자들이 수용할 수 있을 정도의 빠른 속도로 수행되어져야 한다. 기본적으로 $X = Y^{z}\ (mod\ N)$형태의 모듈러 지수승의 반복적인 연산을 수행한다. 안정성(security)을 확보하기 위해 Y, Z, N은 아주 큰 수이어야 하며, 일반적으로 512나 1024비트가 사용되고 있다. 그리고, 이러한 모듈러 지수승은 소프트웨어나 하드웨어 상에서 모듈러 곱셈의 반복적인 연산으로 수행된다.

현재까지 RSA 암호화 시스템의 수행 속도를 증가시키기 위한 많은 알고리즘이 제시되었다. 본 장에서는 기존의 지수승 알고리즘을 고찰하며, 본 논문에서 제안하는 방법을 제시한다.

1. 이진 제곱 곱셈 알고리즘

모듈러 지수 연산을 수행하는 일반적인 방법은 "이진 제곱 곱셈"(binary square and multiply) 알고리즘이다[12]. 이 기법은 각 반복문이 단지 모듈러 곱셈(modular multiplication)과 간단한 이진 결정(binary decision)만을 취하기 때문에 특히 하드웨어 상에서 수행하기에 효율적이다. $M^{e}(mod\ N)$을 수행 시에 암호화 키 e는 $e_{n-1}\, e_{n-2}\, , \ldots, e_{0}$의 이진 형태로 표현되며, 최상위 비트(most significant bit)는 e_{n-1}이다. 이진 제곱 곱셈 알고리즘은 비트 단위로 지수를 검사함으로써 서로 다른 두 가지 형태의 모듈러 곱셈을 수행한다. 이진 제곱 곱셈 알고리즘은 다음과 같다.

```
C = 1;
for(i = n-1; i ≥ 0; i --){
   C = C² (mod N);
   if e_i = 1 then C = C × M (mod N);
}
```

위 알고리즘을 수행시에 필요한 모듈러 곱셈의 연산횟수는 평균 $n/2$이며, 시스톨릭 어레이로 표현시에 $2n$개의 처리기가 필요하다. 이때 요구되는 모듈러 곱셈의 횟수는 정수 e의 이진 표현에서 나타나는 비트 '1'의 개수에 의해 결정된다. 따라서, 정수 e를 다른 수 체계로 표현하여 '0'이 아닌 디지트의 수를 줄임으로써 모듈러 지수승시에 모듈러 곱셈의 수행 횟수를 줄이는 것이 가능하다.

2. 수정된 부호화 디지트 알고리즘

수정된 부호화 디지트(modified signed digit) 표현은 암호화 키 ***E***를 표현함에 있어서 이진 잉여 표현법(binary redundant representation, BRR)을 사용한다. 즉 E는 $\mathbf{B(E)} = b_{n-1}2^{n-1} + b_{n-2}2^{n-2} + \ldots + b_{1}2^{1} + b_{0}$ 로 표현되며 $b_i \in \{0, 1, -1\}$, $i = 0, 1, \cdots, n-1$이다. 즉 이진 표현법에 계수 '-1'을 추가시켜 표현함으로써 '0'이 아닌 디지트 개수를 줄일 수 있다. 다음은 BRR을 적용하여 $C = M^{bi}\ (mod\ N)$ 을 계산하기 위한 모듈러 지수승 알고리즘이다.

```
C = 1;
for(i = n-1; i ≥ 0; i --){
      C = C² (mod N);
      if b_i = 1 then C = C × M (mod N);
      if b_i = -1 then C = C × M⁻¹(mod N);
}
```

위 알고리즘에서 $b_i = 1$ 인 경우에는 메시지 **M**과 모듈러 곱셈을 수행하며, $b_i = -1$ 인 경우에는 메시지 $\mathbf{M}^{-1}$과 모듈러 곱셈 연산을 수행한다. 그리고 $b_i = 0$인 경우에는 모듈러 제곱 연산을 수행한다. 수정된 부호화 디지트 표현은 부스(Booth) 알고리즘을 수행하여 '0' 이 아닌

디지트 수를 최소화하고 유일한 BRR표현을 구할 수 있으나 전체 자리수가 한 자리 더 커지는 경우가 생길 수 있다[13]. 이런 경우에는 후 처리(postprocessing)를 하여 자리수를 일치시켜 주어야 한다. 그리고 모듈러 곱셈 연산 횟수를 $n/3$으로 줄일 수 있으며, 시스톨릭 어레이로 표현시에 $3n/2$개의 처리기(processing element)가 필요하다. 이 때 암호화 키 e가 각각의 처리기에 미리 적재되어 있어야 하며, 모든 메시지의 M^{-1}을 선 처리(preprocessing)를 해서 시스톨릭 어레이 시스템으로 흘려주어야 한다.

3. K-SR 알고리즘

k-SR(string replacement) 알고리즘은 암호화 키 E를 표현함에 있어서 열 치환 표현(string replacement representation)기법을 사용한다. 치환할 크기 K가 정해진 후에 $2 \le i \le k$를 만족하는 i에 대해서 i개의 연속적(consecutive)인 비트 '1'을 2^{i-1}로 변환시킨다. 알고리즘 적용 후에 암호화 키 e는 $f(r-1)f(r-2), \cdots, f(0)$으로 표현된다. $f(r-1)$은 최상위 디지트이며, $f(i) \in (0, 1, 3, 7, \cdots, 2^k-1)$이다. 다음은 $C = M^{f_i} \pmod N$ 을 계산하기 위한 K-SR 모듈러 지수승 알고리즘이다.

```
C = M^{f(r-1)} (mod N);
for(i = r-2; i≥0; i--){
    C = C^2 (mod N);
    if f(i) ≠ 0 then C = C × M^{f(i)} (mod N);
}
```

위의 알고리즘에서 $f(i)$가 0이 아닌 경우에는 미리 계산한 $M^{f(i)}$와 모듈러 곱셈 연산을 수행하며, $f(i)$가 0인 경우에는 이진 제곱 곱셈 알고리즘과 동일하다. K-SR 알고리즘은 모듈러 곱셈 연산의 횟수를 $2^{k-2}/(2^{k-1})$ 로 감소시키며, K가 커질 경우에는 모듈러 곱셈 횟수를 평균 1/4로 줄일 수 있다. 그리고 시스톨릭 어레이로 수행시에 $5n/4$개의 처리기가 필요하며, 모든 선 계산한 값들을 처리기내의 메모리에 보관해야 한다. 그러나 이진 표현에서의 단지 연속적인(consecutive) k개 이내의 비트 '1'만을 K-SR표현으로 변환시킬 수 있는 단점이 있다.

4. SS(l) 알고리즘

Lam과 Hui는 이진 제곱 곱셈 지수연산의 수행 속도를 향상 시키기 위한 $SS(l)$ 알고리즘(string substitution)을 제안하였다[9, 10]. 여기서 l은 선 계산되어지는 지수 e의 최대 길이이며, 선 계산한 메시지의 지수승한 값들을 저장하는 기억장소로서 테이블 H를 사용한다. 그리고 모듈러 지수승을 수행할 때 모듈러 곱셈을 위해 선 계산시에 2^l승 이하의 모든 홀수지수승한 값들을 메모리에 저장시킨다. 그러나 실제 계산시에는 변환후에 단지 필요한 홀수지수승만이 사용되므로, 모듈러 지수승시에 결코 사용되지 않는 선 계산한 값들을 저장함으로써 구현시에 기억 공간 크기를 증가시키는 단점이 있다. 다음은 $x = m^{e[1..i]} \pmod N$을 계산하기 위한 $SS(l)$ 알고리즘이다.

```
x = 1; i = 0;
Loop {
    while(i < n) and (e_{i+1} = 0)
        { i = i+1; x = x^2(mod N); }
    if(i ≥ n) then done and return x as answer.
    j = j+1;
    while(j > n) or (e_j = 0) { j = j-1; }
    a = the value e[(i+1)..j]; b = (a-1)/2;
    while(i < j) { i = i+1; x = x^2(mod N); }
    x = x × H[b] (mod N);
}
```

5. SE(m) 알고리즘

기본적인 이진 제곱 곱셈 알고리즘의 효율은 수행되는 모듈러 곱셈의 횟수에 의해 좌우된다. 즉 지수 e의 이진 표현에서 '0'이 아닌 비트의 수에 달려있다. $SE(m)$ 알고리즘(string exchange)은 선 계산(precomputation)과 비트열 치환(bit string replacement)에 기반한 방법으로서 지수 e를 길이(length) n의 비트열로 간주해서 처리한다. 즉 e의 이진 표현을 고정된 m개의 크기 내에서 불연속적인 비트 '1'의 스트링을 검사하여 홀수(odd number)로 표현 가능한 비트 패턴(1... 1)을 찾아 대치하는 것이다. 기존의 $SS(l)$ 기법과 다른 점은 메시지의 짝수 지수 연산과 전체 홀수 지수 연산을 제외하고, 단지 실제 메시지 지수승에 필요한 홀수 지수 연산만을 처리함으로써 모듈러 지수승시에 선 계산(preprocessing)에 필요한 연산의 수를 반 이하로 줄일 수 있다. 즉 $SE(m)$표현으로 변환 후에 메시지의 선 계산시에 필요로 하는 홀수 지수승을 알 수 있기 때문에 2^m보다 작은 모든 홀수 지수승을 계산할 필요가 없다. 결과적으로 선 처리한 메시지의 지수승의 값들을 저장하는 기억 공간의 크기를 기존의 방법들보다 줄일 수 있다. 한편 e에서 비트 패턴에 포함되지 않는 비트 '0'은 그대로 '0'이 된다. 지수 e의 이진표현을 $SE(m)$표현으로 변환하는 알고리즘은 다음과 같다.

```
i = n;
while(1) {
        while i > 0 and e[i-1] = 0 {
        i = i - 1;
        S(i) = 0; }
        if i = 0 then stop;
        j = i - m;
        while j < 0 or e[j] = 0 {j = j + 1;}
        a = i - j;
        b = the value of (e[i-1] .. e[j]);
        while a > 1 {
                i = i - 1;
                S[i] = 0;
                a = a - 1; }
        S[i-1] = b;
        i = j;
}
```

위의 $SE(m)$ 알고리즘은 지수 e를 최상위 비트에서 최하위 비트 방향(left to right)으로 검사해 나가면서 비트 '0'은 그대로 처리하고 첫번째 비트 '1'을 만나면 이 때부터 최대 m비트 크기내에서 1개의 홀수 디지트를 형성하기 위해 인덱스를 후진하면서 처음으로 검색되는 비트 '1'을 찾는다. 그리고 각 홀수 디지트를 생성한 후에 $m-1$개의 비트 '0'을 남는 자리에 채워 넣는다. 위 알고리즘에 의해 변형된 암호화키 e는 $S_{n-1}S_{n-2}, \ldots, S_0$으로 표현된다. 여기서 S_{n-1}은 최상위 비트이며, 항상 '0' 또는 '1'의 값을 가지며, 나머지 각각의 S_i는 $S_i \in (0, 1, 3, 5, 7, \ldots, 2^{j-1})$, $i<n, j \leq m$이다. 그리고 모듈러 지수승 과정에서 $S_i \neq 0$인 경우 모듈러 곱셈을 수행할 때 사용하기 위해 $M^{S_i}(i \geq 1)$은 미리 계산하여 메모리에 저장시켜 놓는다. $SE(m)$ 알고리즘을 구현시에 다른 윈도우 방법들과 마찬가지로 오버헤드는 선 계산한 값들을 저장하는 메모리 크기에 좌우되므로 2^m보다 작은 모든 홀수 지수승을 계산하는 것이 아니라 변환시에 생성되는 홀수 지수승만을 처리하여 저장한다. 그럼으로써 m이 커짐에 따라 증가하는 선 계산을 위한 기억장치의 크기를 줄일 수 있다. 이용 가능한 메모리가 충분하다고 가정할 경우에, 512비트 지수승시에는 $m=5$일 때 평균 100번의 모듈러 곱셈으로 가장 효율이 좋았으며, 1024비트의 경우에는 $m=6$일 때 177번의 모듈러 곱셈으로 효율이 좋게 나타난다. 한편 모듈러 제곱 연산 횟수는 이진 제곱 곱셈 알

고리즘과 동일하지만, 선 계산시에 추가로 1번의 모듈러 제곱 연산이 더 필요하다. 그리고, 미리 계산되는 메시지의 지수승 개수 l은 이다.

$$l \leq \lfloor (2^m - 1)/2 \rfloor$$

예를 들어 $m=3$일 경우에 미리 계산되어지는 홀수 지수승은 m^3, m^5, m^7이며 각각을 구하는 순서는 다음과 같다. $m \cdot m = m^2$, $m \cdot m^2 = m^3$, $m^3 \cdot m^2 = m^5$, $m^5 \cdot m^2 = m^7$순서로 구할 수 있으며, 이 때 모듈러 제곱 연산은 1번만 수행하고 나머지 연산은 먼저 구한 모듈러 제곱을 연속적으로 적용함으로써 구할 수 있다. 다시 말해서 선 계산시에 먼저 m^2을 구하기 위해 1번의 모듈러 제곱 연산을 필요로 하고, $2^{m-1}-1$의 각각의 홀수 지수승을 구하기 위해 각각 1번씩의 모듈러 곱셈 연산을 수행해야 한다. 선 계산시에 휴리스틱한 방법으로 덧셈 사슬(addition chain) 기법을 사용할 수 있지만 필요한 지수승 뿐만 아니라 많은 중간 계산값들도 메모리에 저장되어 있어야 하기 때문에 비효율적이며 또한 모듈러 제곱과 모듈러 곱셈 연산이 불규칙적으로 사용되는 단점이 있다. 일반적으로 2^m이하의 모든 홀수가 사용된다면 선 계산이 저장되는 기억장소 $T[(i+1)/2-1] = M^i$(i는 홀수)이다.

$SE(m)$ 알고리즘 수행 후에 원래 이진 표현 e와 비교해 보면 매 m비트마다 1번 이상의 모듈러 곱셈을 수행하지 않는다. 그러므로, 전체 n비트의 이진 표현을 $SE(m)$ 표현으로 변환시에 예상되는 모듈러 곱셈의 최대 수는 $\lceil n/m \rceil$이 된다. 여기서 n이 512나 1024와 같이 상당히 큰 수 일 때 예상되는 모듈러 곱셈수는 다음과 같다[10].

$$\lim_{n \to \infty} \frac{\lceil \frac{n}{m} \rceil}{n} = \frac{1}{m+1}$$

다음 표 1은 n이 512비트일 경우의 필요한 모듈러 곱셈수와 메시지의 선 계산수를 나타낸 것이다.

표. n=512일 때 모듈러 곱셈수, 선 계산 수

알고리즘	모듈러 곱셈 수	선 계산 수
이진제곱곱셈 알고리즘	256번	0번
SE(2)적용 알고리즘	170번	≤1번
SE(3)적용 알고리즘	128번	≤3번
SE(4)적용 알고리즘	102번	≤7번
SE(5)적용 알고리즘	85번	≤15번
SE(m)적용 알고리즘	$n/(m+1)$	$\leq \lfloor (2^m-1)/2 \rfloor$

$SE(3)$을 적용한 경우에 이진 표현과 비교해 보면 다음과 같다.

	e_9	e_8	e_7	e_6	e_5	e_4	e_3	e_2	e_1	e_0
이진표현: 939_{10}	1	1	1	0	1	0	1	0	1	1

	S_9	S_8	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0
$SE(3)$표현: 939_{10}	0	0	7	0	0	0	5	0	0	3

이진 제곱 곱셈 알고리즘에 의해 $M^{939} \bmod N$을 계산하기 위해서는 모듈러 제곱 연산 10번, 모듈러 곱셈 연산 7번이 요구되지만, $SE(3)$

알고리즘에 의해서는 모듈러 제곱 연산 10번, 모듈러 곱셈 연산 3번이 요구된다. 위 예제에서 n이 512나 1024정도의 아주 큰 수일 경우에는 모듈러 곱셈의 수를 비교해볼 때 두 표현법에 있어서는 상당한 차이가 있게 된다. 결과적으로 이진 표현을 $SE(m)$표현으로 변형하여 '0' 이 아닌 디지트 개수를 줄임으로써 모듈러 지수 연산의 속도를 높일 수 있다. 위에서 구해진 $SE(m)$표현을 적용해서 $C=M^{S_i}(mod\ N)$을 계산하기 위한 모듈러 지수승 알고리즘은 다음과 같다.

```
C = 1;
for(i = n - 1;i ≥ 0;i - - ){
    C = C^2(mod N);
    if S_i ≠ 0 then C = C × M^{S_i}(mod N) ;
}
```

위 알고리즘은 $S_i = 0$인 경우에는 이진 제곱 곱셈 알고리즘과 동일하게 모듈러 제곱 연산을 수행하며, $S_i \neq 0$이면 미리 계산한 M^{S_i}와 모듈러 곱셈 연산을 수행한다. 즉 알고리즘을 수행하기 전에 $M^3, M^5, \cdots, M^{2i+1}, 1 \leq i \leq m$은 미리 계산하여 메모리에 저장시켜 놓은 후에 그 값을 이용해서 모듈러 곱셈 연산을 한다. 다음 표 2는 위 예의 모듈러 지수승 알고리즘의 적용되는 과정을 순차적으로 나타낸 것이다.

표 2. M939(mod N)을 SE(3)표현으로 모듈러 지수승 알고리즘 적용

$SE(3)$표현 S_i (n=10 경우)		$SE(3)$알고리즘
n-1	0	$C = 1(mod\ N)$
n-2	0	$C = 1(mod\ N)$
n-3	7	$C = 1(mod\ N)$
		$C = M^7(mod\ N)$
n-4	0	$C = M^{14}(mod\ N)$
n-5	0	$C = M^{28}(mod\ N)$
n-6	0	$C = M^{56}(mod\ N)$
n-7	5	$C = M^{112}(mod\ N)$
		$C = M^{112}M^5(mod\ N)$
n-8	0	$C = M^{234}(mod\ N)$
n-9	0	$C = M^{468}(mod\ N)$
n-10	3	$C = M^{936}(mod\ N)$
		$C = M^{936}M^3(mod\ N)$

Ⅲ. 시스톨릭 어레이상에서 $SE(m)$기법 적용

모듈러 지수연산을 병렬로 처리하기 위해서 먼저 $SE(m)$ 알고리즘에 의해서 이진 표현의 정수 e를 $SE(m)$ 형태로 변형한 후, 이를 시스톨릭 어레이 수행에 적합한 시스톨릭 $SE(m)$ 형태로 대치하여 암호화/복호화를 수행한다. 반복적으로 모듈러 제곱과 모듈러 곱셈을 수행하는 모듈러 곱셈기는 김[11]이 제안한 선형 시스톨릭 모듈러 곱셈기(linear systolic modular multiplier)를 사용하기로 한다. 암호화 시스템

에서 e, d, N 은 일반적으로 고정되어 있다. e는 암호화 키, d는 복호화 키, N은 모듈러스(modulus)이다. 또, 처리될 메시지 M은 b개의 블록(즉 $M_1, M_2, M_3, ..., M_b$)으로 분할되어서 암호 시스템에 입력된다고 가정한다(각 메시지의 블록은 모듈러스 N보다 작은 값을 가진다). 일반적으로 순차적으로 수행되는 이진 제곱 곱셈 알고리즘(binary square and multiply)에서 $ei \neq 0$ 이면, 한 개 제어 비트 e_i가 모듈러 제곱과 모듈러 곱셈의 2개 연산을 제어해야 한다. 그러므로, 디지트 레벨의 시스톨릭으로 구현하기 위해서 한 개 디지트(digit)는 한 개의 연산만을 수행하도록 간단히 변형을 가한다. $SE(m)$표현을 시스톨릭 $SE(m)$표현으로 변환하는 알고리즘은 다음과 같다.

```
i = 0;
j = 0;
while i < n {
    S_j = s_i;
    if s_i = 0 then { j = j + 1;}
    if s_i ≠ 0 then { S_{j+1} = 0; j = j + 2; }
    i = i + 1;
}
```

위 변환 알고리즘은 최하위 디지트부터 $SE(m)$표현의 s_i를 읽어들여서 시스톨릭 표현으로 변환시킨다. 즉 각 '0' 디지트는 시스톨릭 표현에서 '0'으로 변환되고, 각 '0'이 아닌 디지트는 시스톨릭 표현에서 2개 디지트로 변환된다. 즉 원래의 디지트와 상위 디지트에 '0'을 취한 두 개 디지트로 표현된다. 시스톨릭 표현에서 각 '0' 디지트는 제곱 연산, 각 '0'이 아닌 디지트는 모듈러 곱셈 연산을 수행한다. 위의 시스톨릭 표현은 기본적인 '이진 제곱, 곱셈' 알고리즘에도 적용 가능하다. 다음 표 3은 $M^{939}(mod\ N)$을 시스톨릭 표현으로 적용한 것이다.

표 3. $M^{939}(mod\ N)$을 시스톨릭 $SE(3)$표현으로 모듈러 지수승 알고리즘 적용

시스톨릭 $SE(3)$표현		시스톨릭 $SE(3)$ 알고리즘
n-1	0	$C = 1(mod\ N)$
n-2	0	$C = 1(mod\ N)$
n-3	0	$C = 1(mod\ N)$
n-4	7	$C = M^7(mod\ N)$
n-5	0	$C = M^{14}(mod\ N)$
n-6	0	$C = M^{28}(mod\ N)$
n-7	0	$C = M^{56}(mod\ N)$
n-8	0	$C = M^{112}(mod\ N)$
n-9	5	$C = M^{112}M^5(mod\ N)$
n-10	0	$C = M^{234}(mod\ N)$
n-11	0	$C = M^{468}(mod\ N)$
n-12	0	$C = M^{936}(mod\ N)$
n-13	3	$C = M^{936}M^3(mod\ N)$

위 시스톨릭 표현을 선형 시스톨릭 어레이(linear systolic array)로 구현할 시에 한 개 PE(processing element)의 구조는 그림 1과 같다. 각 PE는 크게 멀티플렉서(Mux) 3개와 레지스터(Reg) 1개, 그리고 모듈러 곱셈기(MMM)로 구성된다. 각 PE로 입력되는 값은 메시지 $Mi(i=1,...,b)$, 암호/복호화 키 S_i, 그리고 선 계산한 값$M_i^{S_j}(mod\ N)$이다. 그림 2의 선형 시스톨릭 어레이 구조에서 전체 어레이는 $M^e(mod\ N)$을 계산하기 위해 공통 클락(common clock)과 동기화된다. 기존에 제안된 시스톨릭 어레이 표현에서는 암호화 키가 각 처리기에 선 적재되어 있어야 한다. 이런 형태의 어레이에서 선 적재는 처리기의 개수가 많아져서 분할 방법을 적용할 경우에 장애요인이 된다. 그러나 본 논문에서는 암호화 키 값을 하나의 제어 신호 Ctl을 사용하여 외부에서 입력되도록 한다. 그럼으로써 각각의 PE는 $Ctl=1$일 때 그 PE에 입력된 S_i값을 해당 PE가 레지스터에 보관하여 계속 사용하고, $Ctl=0$일때는 이미 저장된 S_i값을 사용하며, 입력되는 S_i값을 링크 방향에 따라 이웃한 PE로 흘려 보낸다. 이 때 흐르는 S_i값이 적절한 PE에서 이용되도록 하기 위하여 Ctl신호는 1개의 지연(delay)을 가지고 흐르도록 한다. 그리고 $M^{S_j}(mod\ N)$ 값은 M_{in}값과 동일하게 PE들 사이를 경유하여 마지막 PE까지 흘러간다. 입력되는 S_i값에 따라서 즉 아래쪽의 멀티플렉서에서 $S_j=0$이면 입력된 메시지를 그대로 모듈러 제곱을 수행해서 다음 PE로 보내고, 만약 $S_j\neq 0$이면 위쪽의 멀티플렉서에서 해당되는 미리 계산한 $M^{S_j}(mod\ N)$을 가져와서 모듈러 곱셈을 수행하고 다음 PE로 보내게 된다. 이 때 위에서 설명한 바와 같이 각 PE는 Ctl신호에 따라서 적절한 S_i값을 선택하여서 계산을 수행하게 된다. 전체적으로 볼 때 b개의 메시지가 파이프라인(pipeline) 방식으로 처리된다. 공개 키 암호화 시스템을 위한 선형 시스톨릭 어레이는 그림 2와 같다.

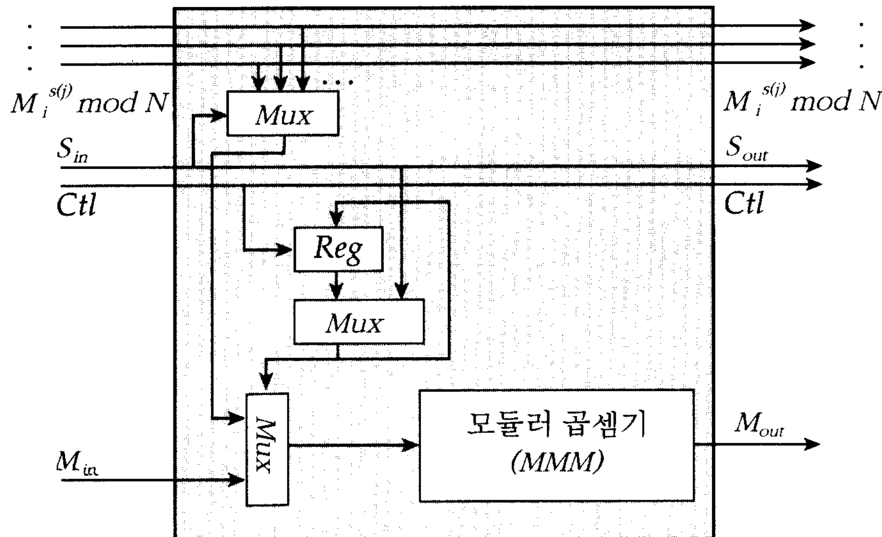


그림 1. PE구조

첫 번째 입력 메시지가 어레이로 흘러 들어간 후에 출력에 첫 결과가 처리되어 나오는데는 $n(m+2)/(m+1)$ 클락이 소요된다. 그리고 시스톨릭 어레이의 면적(area)을 시스톨릭 어레이를 이루는 전체 처리기의 개수로 정의하면 time×area는 $O(n^2(m+2)^2/(m+1)^2)$이다. Zhang[6]이 제안한 시스톨릭 어레이 구조와 비교해 보면, 이진 제곱 곱셈 알고리즘을 선형 시스톨릭 어레이로 구현시에 $2n$개의 PE가 요구되며 또한 두 종류의 PE가 필요하다. 즉 홀수번째 PE는 모듈러 제곱을 수행하고, 짝수번째 PE는 모듈러 곱셈만을 수행하며, 암호화키 e는 다른 입력값들과 달리 다음 PE로 흐르지 않고 단지 짝수번째 PE에 정적으로 고정되어 있다. 첫 번째 결과가 나오는데 $2n$ 클락이 소요되며, time×area는 $O(4n^2)$이다. 본 논문에서 제안한 방법으로 비교할 경우에 $m=5$일 때 $5n/6$개의 PE를 줄이게 된다. 그리고 수정된 부호화 디지트 알고리즘을 선형 시스톨릭 어레이로 구현시에 $3n/2$개의 PE가 요구되며, 첫 번째 결과가 출력되는데 $3n/2$ 클락, time×area는 $O(9n^2/4)$이며, 이진 제곱 곱셈 알고리즘과 동일하게 e는 정적으로 각 PE에 고정 할당되어 있다. 본 논문에서 제안한 방법과 비교하면 $n/3$개의 PE를 감소하게 된다.

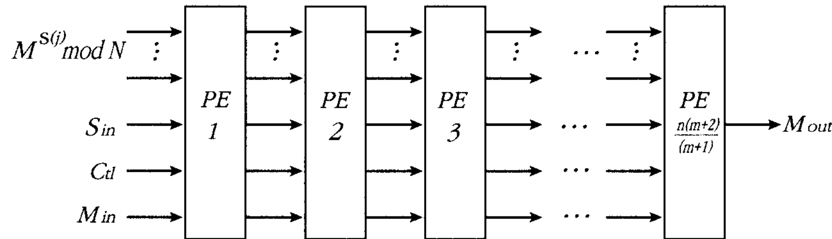


그림 2. SE(m)알고리즘에 기반한 선형 시스톨릭 어레이

다음의 그림 3은 시스톨릭 $SE(3)$을 이용한 선형 시스톨릭 어레이의 적용 예이다. 암호화키 e가 십진수로 $885(1101110101_2)$일 때, 시스톨릭 $SE(3)$표현으로는 "0300070005" 가 되며 S_9 는 최상위 비트 '0' 이다. 그리고, Ctl신호는 PE갯수 만큼 어레이로 흘러 들어간다.

e가 512비트 이상의 데이터일 경우에 제한된 크기의 칩내에 $n(m+2)/(m+1)$개의 처리기를 가지는 VLSI칩의 설계는 불가능하다. 이런 경우에는 분할(partitioning) 방법을 사용하여 큰 크기의 어레이를 여러 개의 그룹으로 분할하여, 각 그룹을 작은 크기의 어레이로 사상하면 된다[11, 14]. 이 때 한 그룹내의 모든 처리기는 동시에 수행되며, 각 그룹간의 수행은 의존관계에 따라서 순차적으로 이루어 진다. 만약 그룹간의 사이클이 존재한다면 그룹의 수행 순서를 결정하지 못하므로 올바른 수행이 되지 않는다. 그리고 한 그룹 수행 후에 생성되는 중간 결과값은 프로세서 어레이 외부의 FIFO버퍼에 저장된다. i번째 그룹의 수행이 끝난 후에 다음 $i+1$번째 그룹의 수행이 시작되므로 i번째 그룹의 출력값들은 $i+1$그룹이 시작될 때 까지 버퍼에 저장되었다가 적절하게 입력되어야 한다.

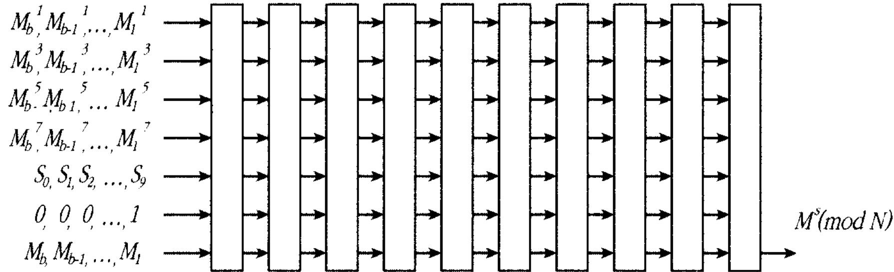


그림 3. 선형 시스톨릭 어레이(m=3)

Ⅳ. 비교 분석

표 4는 n비트의 모듈러 지수연산시에 요구되는 모듈러 곱셈과 제곱의 수에 의하여 각 알고리즘을 비교 분석하고, 선형 시스톨릭 어레이로 구현시에 필요한 처리기 수를 제시한다(n은 일반적으로 512나 1024비트이다). 표에서는 메시지를 미리 계산하는 연산과 실제 연산의 두 부분으로 나누어 비교한다.

표4. 알고리즘 비교 분석

분석 / 알고리즘	계 산(computation)			선 계산(precomputation)			처리기 수
	Worst Mult	Average Mult	Square	Values	Mult	Square	
이진제곱곱셈	n	$n/2$	n	None	None	None	$2n$
수정된 부호화 디지트	$n/2$	$n/3$	n	T-1	None	None	$3n/2$
K-SR	$\lceil n/2 \rceil$	$n/4$	n	$T^3, T^7, \ldots T^{2K}$	k-1	k-1	$5n/4$
시스톨릭 SE(m)	$\lceil n/m \rceil$	$n/(m+1)$	n	$T^3, T^5, \ldots T^{2m-1}$	$\leq 2^{m-1}-1$	1	$n(m+2)/(m+1)$

다음 표 5에서는 각 알고리즘을 실행시에 수행되는 평균 곱셈 수와 시스톨릭 어레이를 구성시에 요구되는 처리기의 수를 $SE(m)$ 알고리즘을 기준으로 해서 각각 비교분석해 보았다. 수정된 부호화 디지트 알고리즘과 비교해 보면, 평균 곱셈수에서는 57% 그리고 처리기 수에서는 24% 정도로 더 효율이 좋게 나타나고 있다.

표 6과 표 7에서는 각각 512와 1024비트의 모듈러 지수승에 요구되는 전체 모듈러 곱셈 수를 분석해 보았다. 여기서 전체 모듈러 곱셈수는 선 계산시에 최대로 생길 수 있는 모듈러 곱셈까지 고려한 것이다. 표 6에서 512비트의 경우에는 m=5일때 평균 100번(85+15)의 모듈러 곱셈으로 가장 효율이 높았으며, 표 7에서 1024비트의 경우에는 m=6일 때 177번

표 5. n=1024일 때 평균 곱셈 수와 처리기 수의 비교

알고리즘 / 성능	이진제곱곱셈 (n_1)	수정된 부호화 디지트 (n_1)	5-SR (n_1)	시스톨릭 $SE(6)$ (n_2)
평균곱셈 수	512	341	256	146
$(n_1-n_2)/n_1$	71%	57%	43%	0%
처리기 수	2048	1536	1280	1170
$(n_1-n_2)/n_1$	43%	24%	9%	0%

표 6. SE(m)을 이용한 512비트 지수승에 요구되는 모듈러 곱셈수

m	Worst case	Average case
1	512	256
2	257	171
3	174	131
4	135	109
5	118	100
6	117	104
7	137	127
8	191	183
9	274	266
10	563	557

표 6. SE(m)을 이용한 1024비트 지수승에 요구되는 모듈러 곱셈수

m	Worst case	Average case
1	1024	512
2	513	342
3	345	259
4	263	211
5	220	185
6	204	177
7	210	191
8	255	240
9	369	357
10	614	604

(146+31)번의 모듈러 곱셈으로 효율이 좋게 나타나고 있다. 그리고 최대 효율치 m을 넘는 경우에는 최악의 경우와 평균의 경우에 두 값의 차이가 점점 줄어드는 것을 알 수 있다.

V. 결 론

암호화 시스템에서 모듈러 지수승은 512비트 이상의 아주 큰 수의 반복적인 모듈러 곱셈을 필요로 한다. 본 논문에서는 모듈러 지수 연산에서 반복되는 모듈러 곱셈 횟수를 줄이기 위하여 시스톨릭 $SE(m)$ 알고리즘을 제시하였으며 병렬 연산을 위한 선형 시스톨릭 어레이를 제안하였다. 모듈러 지수 연산을 수행시에 $n/(m+1)$개의 모듈러 곱셈 연산을 필요로 하며, $n(m+2)/(m+1)$개의 처리기를 사용한다. 그리고, 선 계산시에 단지 필요한 홀수 지수승만을 처리함으로써 요구되는 기억 공간의 크기를 줄였으며, 시스톨릭 어레이로 구현시에 암호/복호화 키의 선적재 문제를 해결하였고, 또한 처리기 내부에 단지 암호/복호화 키를 저장할 기억 장소만을 사용하였다. 미리 계산할 비트열의 크기 m이 증가함에 따라 연산의 수행 속도는 증가하지만 비례하여 선 계산량

과 보조 기억 공간의 크기도 증가하므로, 수행 시간과 기억 공간을 고려하여 적합한 m의 크기를 선택해야 한다. 본 논문에서 제안한 모듈러 지수승을 위한 시스톨릭 어레이 모델은 선형 시스톨릭 어레이로 구현될 수 있으며, 다른 제안된 시스톨릭 어레이 기법보다 더 적은 수의 처리기를 사용한다. 그리고 처리기 개수를 줄이기 위하여 분할 방법을 이용한다면 제한된 크기내에 모든처리기를 집적할 수 있다.

모듈러 지수 연산은 아주 큰 수를 반복적으로 처리하기 때문에 일반 범용 컴퓨터에서는 상당히 느린 속도로 수행된다. 수행 속도를 높이기 위해서는 모듈러 지수승을 위한 하드웨어 칩의 구현이 필수적이다. 앞으로의 연구과제는 본 논문에서 제안한 방법과 Zhang이 제안한 방법을 VHDL을 사용하여 하드웨어로 구현함으로써 실제 하드웨어 레벨에서 여러가지 성능을 비교해 보는 것이다.

참고문헌

[1] W. Diffie, M. Hellman, "New Directions in Cryptography," *IEEE* Trans. on Info. Theory, vol. IT-22(6) pp.644-654, 1976.

[2] Rivest, R. L., Shamir, A. and Adleman, L., "A method for obtaining digital signatures and public-key cryptosystems," *Communication of the ACM*, 21, 120- 126. 1978.

[3] C. D. Walter, " Systolic modular multiplication," *IEEE Trans. Computers*, 42(3), pp. 376-379, 1993.

[4] P. L. Montgomery, "Modular Multiplication Without Trial Division," *Mathemat. of Computat.* vol. 44, pp. 519-521, 1985.

[5] Jedwab, J. and Mitchell, C. J., "Minimum weight modified signed-digit representaions and fast exponentiation," *Electronics Letters*, 25(17), pp. 1171-1172, 1989.

[6] Zhang, C. N., " An improved binary algorithm for RSA," *Computer Math. Applic.*, 25(6), pp. 15-24, 1993.

[7] Gollmann, D., Han Yongfei and Mitchell, C., "Redundant integer representations and fast exponentiation," *Designs, Codes and Cryptography*, 7, pp. 135-151, 1996.

[8] Y. Han. D. Gollmann. and C. J. Mitchell, "Fast modular exponentiation for RSA on systolic arrays," International Journal of Computer Mathematics, 61, 1996.

[9] Hui, L. C. K. and Lam, K. Y., "Fast square-and-multiply exponentiation for RSA," Electronics Letter, 30(17), pp.1396-1397, 1994.

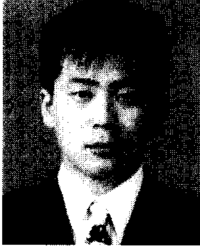
[10] Lam, K. Y. and Hui, L. C. K., "Efficiency of SS(l) square and multiply exponentiation algorithms," Electronics Letter, 30(25), pp. 2115-2116, 1994.

[11] 김 현철, "모듈러 곱셈을 위한 선형 고정-크기 시스톨릭 어레이 설계," 석사학위논문, 경북대학교, 1997.

[12] Knuth, D. E., " The art of computer programming," Volume 2: seminumerical algorithms. Addison-Wesley, Reading, Mass., 2nd edition, 1981.

[13] A. D. Booth. " A signed binary multiplication technique," Quarterly Journal of Mechanics and Applied Mathematics, 4:236-240, 1951.

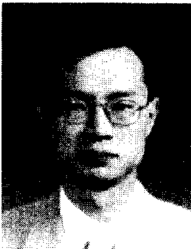
□ 著者紹介



이 건 직

1996년 월 대구대학교 전자계산학과 졸업(공학사)
1998년 월 경북대학교 대학원 컴퓨터공학과 졸업(공학석사)
1982년 12월 ~현재 경북대학교 대학원 컴퓨터공학과 졸업(공학석사)

※ 주관심분야 : 보안/암호, 배열 처리기 설계



허 영 준

1993년 월 대구대학교 전자계산학과 졸업(공학사)
1996년 월 경북대학교 대학원 컴퓨터공학과 졸업(공학석사)
1996년 월 ~현재 경북대학교 대학원 컴퓨터공학과 박사과정

※ 주관심분야 : 병렬처리, 배열 처리기 설계, 보안/암호

유 기 영

1976년 월 경북대학교 수학교육학과 졸업(이학사)
1978년 월 한국과학기술원 전산학과 졸업(공학석사)
1996년 월 미국 Rensselaer Polytechnic Institute 졸업(이학박사)
1996년 월 ~현재 경북대학교 컴퓨터공학과에 재직

※ 주관심분야 : 정보보호/보안, 병렬처리 DSP array processor 설계, 병렬 컴파일러 등임.