정보학연구 제3권 제1호 2000년 3월(pp.61-72)
The Studies of Information Technology, Vol.3, No.1

비트 슬라이스 모듈러 곱셈 알고리즘

류동렬*/조경록**/유영갑***

요 약

본 논문에서는 RSA 공개키 암호시스템에서 암호의 안전성을 위하여 증가되는 암호키(key)의 비트 크기에 대응한 내부 연산기 설계를 효율적으로 할 수 있는 bit-slice형 모듈러 곱셈 알고리즘을 제안하였고, 제안된 알고리즘에 따른 모듈러 곱셈기를 FPGA칩을 이용하여 구현함으로써 제안된 알고리즘의 동작을 검증하였다. 제안된 bit-slice형 모듈러 곱셈 알고리즘은 Walter 알고리즘을 수정하여 도출하였으며, 구현된 모듈러 곱셈기는 bit-slice 구조로 되어 암호키(key)의 비트 확장에 대응한 모듈러 곱셈기의 오퍼랜드 비트 확장이 용이하며, 표준 하드웨어 기술언어(VHDL)로 모델링 하여 전용 하드웨어로 설계되는 RSA 공개키 암호 시스템의 구현에 응용될 수 있도록 하였다.

Ⅰ. 서론

네트워크 환경의 보편화 경향에 따라 정보 보호를 위한 암호화의 필요성이 더욱 강조되고 있다. 여러 암호 시스템 중에서 공개키 암호 시스템은 암호키(key) 분배가 간편하여 네트워크 환경에 적합하다고 여겨지고 있다.[1][2] RSA 공개키 암호 시스템은 소인수 분해의 계산 복잡도가 암호키(key)의 비트 크기에 따라 급격하게 증가하여 소인수 분해된 숫자를 암호에 사용할 경우 암호화 키(key)는 공개되어도 그 해독키(key)를 찾아 내는 것이 매우 어렵다는 사실에 기초한 방법이다.[2][3] 실제로 암호키(key)가 작은 숫자인 경우에는 공격의 위험성이 있으나 키를 200자리 이상의 숫자로 할 경우에는 보안 능력이 있었으나 최근 컴퓨터의 연산능력의 향상으로 암호키(key)를 300자리 이상의 숫자로 사용하여야 할 필요가 있다.[3]

RSA 공개키 암호 시스템에서 공격에 대한 대처 방법으로는 암호키(key)의 크기를 확장하여 공격의 연산 복잡도를 증가시켜 공격의 위험성을 낮추는 방법을 사용하고 있다.[3] 따라서 RSA 공개키 암호 시스템은 매우 큰 수에 대한 모듈러 지수승 연산이 요구된다. 모듈로 지수승 연산은 모듈로 곱셈과 모듈로 제곱의 반복으로 구현되므로 모듈러 곱셈기와 모듈러 제곱 연산기의 설계가 RSA 공개키 암호시스템에서 큰 비중을 차지하게 된다.[4][6][7]

1978년에 RSA 공개키 암호시스템이 제안된 이후 전용 하드웨어로 RSA 공개키 암호시스템을 구현하기 위한 많은 연구가 진행되어 왔다.[8][9][10][11][12][14] 이러한 연구의 주요 목표는 매우 큰 정수에 대한 연산 알고리즘의 고속화가 주요 목표였다. 그러나, 이러한 연구에서는 암호키(key)의 비트 크기가 확장될 경우의 재설계 문제를 고려하지 않았다.

* 충북대학교 정보통신공학과
** 충북대학교 정보통신공학과 부교수
*** 충북대학교 정보통신공학과 교수

본 논문에서는 RSA 공개키 암호 시스템의 핵심 연산인 모듈러 곱셈이 bit-slice 기법을 통하여 수행할 수 있도록 하여 암호키(key)의 비트 확장에 따른 내부 연산기의 오퍼랜드 확장을 위한 재설계를 고려한 수정된 모듈러 곱셈 알고리즘을 제안하였다.

본 논문의 Ⅱ장에서는 모듈러 곱셈 알고리즘에 대하여 기술하고, Ⅲ장에서는 제안된 bit-slice형 모듈러 곱셈 알고리즘 및 FPGA를 이용한 제안된 알고리즘의 하드웨어 구현 및 동작검증에 관하여 기술하고, Ⅳ장에서 결론을 맺도록 한다.

Ⅱ. 모듈러 곱셈 알고리즘

2.1 제안하는 모듈러 곱셈 알고리즘

모듈러 곱셈(1)을 효율적으로 수행할 수 있는 알고리즘은 Montgomery에 의하여 제안되었다.[5]

$$C = A \cdot B \pmod{N} \qquad (1)$$

모듈러 곱셈을 수행하는 관용적인 방법은 두 수의 곱(A · B)에 다시 나눗셈연산을 이용한 모듈러 리덕션(modular reduction)연산을 하게된다. 따라서 관용적인 방법에서는 큰 수에 대하여 나눗셈 연산을 해야 하는 부담이 있다.[9] 이를 해결한 것이 Montgomery 모듈러 리덕션 알고리즘으로 모듈러 연산에 필요한 나눗셈에 대한 부담을 경감시키기 위해 수체계의 변환을 통하여 모듈러 리덕션이 쉽게 되는 수체계에서 연산을 한 후 다시 원래의 수체계로 환원시키는 방법을 이용한다.[10]

Montgomery multiplication 알고리즘에 다정도 모듈러 곱셈(multiple precision modular multiplication)의 적용을 시도한 것이 C. D Walter에 의해 제안된 walter 알고리즘이다.[8] (그림 1)

Walter 알고리즘(그림 1)에서는 연산 ①,②를 통하여 Montgomery 모듈러 리덕션 알고리즘과 다정도(multiple precision) 곱셈이 동시에 수행된다.[9]

본 논문에서는 binary 수체계에서 다정도 곱셈연산을 수행하며, Walter의 알고리즘으로부터

INPUT : *integer* $N=[N_{n-1}\cdots N_1, N_0]_b$, $X=[x_{n-1}\cdots x_1x_0]_b$, $Y=[y_{n-1}\cdots y_1y_0]_b$
with $0 \le X, Y < N$, $R=b^n$, *with* $\gcd(N, b)=1$, and $N=-N^{-1} \bmod b$
MMULT(X, Y, N, R)
  $A=0$; $(A=[a_{n-1}\cdots a_1a_0]_b)$
  for i from 0 *to* $(n-1)$ *do*
    $u_i=(a_0+x_i\cdot y_0)N \bmod b$; ----①
    $A=A+x_iY+u_iN)/b$; ----②
  end for;
  if $A \ge N$ *then* $A=A-N$;
  end if;
return(A)

그림 1. Walter 알고리즘
Fig. 1. Walter's algorithm

암호키(key)의 비트 크기 증가에 따라 암호시스템의 내부 연산기들이 오퍼랜드 비트 확장성을 갖도록 bit-slice형 모듈러 곱셈알고리즘을 다음과 같이 도출하였다.(그림 2)

두 수의 일반적인 곱셈이며 R^{-1}를 곱하는 과정은 모듈러 리덕션 연산에 해당한다.

모듈러스 배수(modulus multiple U_j)는 모듈러스(M)에 일정한 수를 곱하여 최하위 비트가

CONDITION : *multiplicand* and *multiplier is* $2n$ *bit, slice is* n *bit.*
INPUT : *integer* $M[n_{2n-1}\cdots n_1n_0]_2$, $X=[x_{2n-1}\cdots x_1x_0]_2$, $Y=[y_{2n-1}\cdots y_1y_0]_2$,
with $0\le X, Y<M$, $R=2$, *with* $\gcd(M,2)=1$, and $M'=-M^{-1}$ *mod* 2.
OUTPUT : XYR^{-1} *mod* M.
$A_i = 0$, $A_i=[a_i(n-1)\cdots a_i(1)a_i(0)]_2$
for i from 0 *to* 3 *do* ----------①′
if $i=0$ *then* $M_i=M[2n-1:n]$; $X_i=X[2n-1:n]$; $Y_i=Y[2n-1:n]$;
else if $i=1$ *then* $M_i=M[\frac{3n}{2}-1:\frac{n}{2}]$; $X_i=X[2n-1:n]$; $Y_i=Y[n-1:0]$;
else if $i=2$ *then* $M_i=M[\frac{3n}{2}-1:\frac{n}{2}]$; $X_i=X[n-1:0]$; $Y_i=Y[2n-1:n]$;
else if $i=3$ *then* $M_i=M[n-1:0]$; $X_i=X[n-1:0]$; $Y_i=Y[n-1:0]$;
end if;
for j from 0 *to* $n-1$ *do* ----------②′
$U_j=(A_i[0]+X_i[j]\cdot Y_i[0])$ *mod* 2; ----------③′
$A_i=(A_i+X_i[j]\cdot Y_i+U_j\cdot M_i)/2$; ----------④′
end for;
if $A_i \ge M_i$ *then* $A_i=A_i-M_i$
end if;
$A=A+A_i$; ----------⑤′
end for;
return(A)

그림 2.제안하는 bit-slice형 모듈러 곱셈 알고리즘
Fig. 2. Proposed bit-sliced modular multiplication algorithm

제안하는 비트 슬라이스형 모듈러 곱셈 알고리즘은 인덱스가 1차원 공간인 Walter 알고리즘으로부터 인덱스가 2차원 공간인 알고리즘으로 확장시켜 연산의 병렬성을 이용하고자 한 알고리즘이다. 그림 2의 인덱스 i는 4개의 슬라이스 중에 하나를 지정하고 인덱스 j는 각 슬라이스에서의 비트 스캔(bit scan)을 의미한다.

제안 알고리즘에서는 임의의 수 x에 대한 M-residue를 xR mod M으로 정의하며, 연산결과는 M-residue로 변환된 X, Y에 대한 모듈러 곱셈 $X\cdot Y\cdot R^{-1}$ mod M이다. 여기서 $X\cdot Y$는 '0'이 되도록 하여 R로 나누어 떨어지도록 한다. 이러한 모듈러스 배수의 필요성은 나눗셈 연산을 쉬프트 연산으로 대체 할 수 있는 잇점을 제공한다.

Walter 알고리즘에서는 먼저 모듈러스(M)보다 크고 모듈러스와 서로소(gcd(M,R)=0)인 R을 정의하는데 비해 제안 알고리즘에서는 mod R과 div R이 용이하도록 R=2를 사용한다. 따라서 mod 2 연산은 최하위 비트만을 검사하면 되고 div 2의 연산은 쉬프트 동작으로 대체할 수 있다.

제안 알고리즘에서는 Walter 알고리즘의 장점

인 연산의 병렬성을 이용하여 동일한 구조의 n 비트 입력에 대한 슬라이스를 병렬 동작시켜 2n 비트 입력 오퍼랜드에 대한 연산을 수행하도록 한다.

두 입력 X, Y가 $2n$ 비트이고 기본 슬라이스가 n 비트에 대한 연산을 수행한다면, Walter 알고리즘은 $(8n^2+2n)$회의 단정도 곱셈연산을 수행하므로 $O(n^2)$의 시간 복잡도를 갖는다. 한편, 제안된 알고리즘에서는 $4\times(2(n)^2+(n))=(8n^2+4n)$회의 단정도 곱셈연산이 수행되어 알고리즘의 시간 복잡도는 $O(n^2)$으로써 Walter 알고리즘과 동일한 시간복잡도를 갖는다.

2.2 제안 알고리즘의 확장성

제안 알고리즘은 n 비트 모듈러 곱셈기를 기본 슬라이스로 이용하여 4개의 기본 슬라이스를 병렬 연결하여 2n 비트 모듈러 곱셈연산을 수행한다.

수식 (2)와 같이 일반 곱셈연산을 비트 슬라이스 구조로 연산하는 방법에서는 두 입력 X, Y를 상위 비트군(X_H, Y_H)과 하위 비트군(X_L, Y_L)으로 분리하여 4개의 부분곱을 구하여 각각의 부분곱에 대한 캐리 전파 덧셈을 통하여 결과값을 얻는다.

$$\begin{aligned} P &= X\times Y=(X_H\cdot X_L)\times(Y_H\cdot Y_L)\\ &= X_H\times Y_H+X_H\times Y_L+X_L\times Y_H+X_L\times Y_L\\ &= P_{HH}+P_{HL}+P_{LH}+P_{LL} \end{aligned}$$

$X=X_H\cdot X_L$이며 이때, $\cdot$는 concatenation 이다. (2)

제안 알고리즘은 일반 곱셈연산의 비트 슬라이스 방법(수식 (2))과 같이 두 입력 오퍼랜드를 상·하위 비트군 나누어 4개의 모듈러 곱셈을 수행하여 4개의 부분곱을 구한 후 4개의 부분곱에 대한 캐리 전파 덧셈 결과로써 모듈러 곱셈의 최종 결과값을 얻는다. 그러나, 일반 곱셈연산과는 달리 모듈러 곱셈연산에서는 모듈러 감소 연산을 수행하여야 하므로 모듈러스(modulus)에 대한 고려가 필요하다.

$$\begin{aligned} P &= X\times Y=(X_H\cdot X_L)\times(Y_H\cdot Y_L) \mod M\\ &= (X_H\times Y_H \mod M_1)+(X_H\times Y_L \mod M_2)\\ &\quad +(X_L\times Y_H \mod M_3)+(X_L\times Y_L \mod M_4)\\ &= P_{HH}+P_{HL}+P_{LH}+P_{LL} \end{aligned}$$

(3)

수식 (3)에서 모듈러스(M)은 M_1, M_2, M_3, M_4로 분할 하였다. 모듈러스(M)의 분할은 2n 비트의 두 입력 오퍼랜드 X, Y가 모두 상위 비트군(X_H, Y_H)일 경우 모듈러스 $(M_1)=M[2n-1:n]$이고, (X_H, Y_L)인 경우와 X_L, Y_H인 경우 모듈러스 (M_2)와 모듈러스 (M_3)는 $M[\frac{3n}{2}-1:\frac{n}{2}]$이고, ($X_L$, Y_L)인 경우 모듈러스 (M_4)는 $M[n-1:0]$이다.

제안하는 알고리즘(그림 2)의 순환문 ①′에서 4개의 n 비트 모듈러 곱셈으로 분할하고 순환문 ②′에서는 Walter 알고리즘을 이용하여 각 모듈러 곱셈연산(③′, ④′)을 수행하며, 연산 ⑤′에서는 4개의 모듈러 곱셈 연산결과를 더하여 2n 바트의 모듈러 곱셈 결과를 얻는다.

제안하는 알고리즘은 4개의 n 비트 모듈러 곱셈연산을 지시하는 인덱스 i와, n 비트 모듈러 곱셈연산을 수행하기 위한 인덱스 j로 표현하여 2n 비트 모듈러 곱셈연산을 4개의 n 비트 모듈러 곱셈연산으로 분할시킨다. 4n 비트 모듈러

곱셈연산을 수행하기 위해서는 n 비트 모듈러 곱셈연산으로 구성되는 2n 비트 모듈러 곱셈연산을 4번 수행하여 4n 비트 모듈러 곱셈연산을 수행할 수 있다. n 비트 모듈러 곱셈를 이용하여 2n 비트 모듈러 곱셈이 수행하려면 4개의 n 비트 모듈러 곱셈이 필요하고, 4n 비트 모듈러 곱셈연산을 수행하려면 16개의 n 비트 모듈러 곱셈이 필요하다. 즉, 제안하는 알고리즘에서는 입력 오퍼랜드 X,Y의 비트 크기가 증가 할 수록 소요되는 기본 비트 슬라이스의 거듭제곱으로 증가된다.

Ⅲ. Bit-slice형 모듈러 곱셈기의 구현

본 논문에서는 제안된 알고리즘을 구현하기 위해 64× 2^n 비트 크기로 암호키(key)가 확장되는 것을 가정하여 64 비트 오퍼랜드를 위한 bit-slice형 모듈러 곱셈기를 기본으로 하였으며 ALTERA사의 FPGA(FLEX10K100ARC240-3)칩을 이용한 하드웨어 제작을 통하여 제안한 알고리즘의 동작을 검증하였다.

3.1 64 비트 모듈러 곱셈기 slice의 구현

곱셈기의 구현을 위한 slice의 회로를 보기로 한다. 먼저 기본 process element(PE)를 설계하고 이것을 이용하여 모듈러 곱셈기 slice를 만든다. PE는 제안한 알고리즘의 연산과정에서 슬라이스 비트 크기만큼 반복하는 내부 loop ②′의 연산을 수행한다. PE의 비트 크기는 연산 ($A_i=(A_i+X_i[j]\cdot Y_i+U_j\cdot M_i)/2$)에 의해 n의 비트 크기에 의존한다.

본 논문에서는 64×2^n 비트 단위의 암호키(key) 확장을 가정하므로 64 비트 bit-slice형 모듈러 곱셈기에 필요한 64 비트 오퍼랜드에 대한 PE를 설계하였다. 제안한 알고리즘(그림 2)의 내부 loop ②′에서 ($A_i[j]$, Y_i, M_i)이 각각 64 비트에 대한 연산이므로 그림 3에서와 같이 PE는 64 비트의 입력($A_i[63:0]$, $Y_i[63:0]$, $M_i[63:0]$)신호와 1 비트 입력신호인 x_i, 그리고 64 비트의 출력 신호($A_{i+1}[63:0]$)를 갖는다.

제안한 알고리즘에서 R의 선택은 gcd(M,R)=1, $RR^{-1}-MM'=0$ 을 만족하는 범위 내에서 한다. 이때, R = 2 로 선택하면 그림 2의 순환문 ②′에서의 나눗셈 연산은 쉬프트 연산으로 대신 할 수 있고, mod 연산은 최하위 비트 선택으로 대신 할 수 있어 연산의 복잡도를 낮출 수 있다.

PE는 모듈러 곱셈기의 주요 연산부 이므로 PE의 최적화 정도가 모듈러 곱셈기의 지연시간 및 회로크기에 큰 영향을 준다. 본 논문에서는 PE의 최적화는 다루지 않고 동작 함수만을 고려하여 모듈러 곱셈기 설계에 사용하였다.

64 비트 PE를 64개를 반복 사용함으로써 구현되는 64 비트 bit-slice형 모듈러 곱셈기 slice를 그림 4에 보였다.

VHDL 시뮬레이션 툴을 이용한 모의 동작 실험을 통하여 64 비트 bit-slice형 모듈러 곱셈기의 동작을 검증하였다. 그림 5는 M=d'7, n=(64), R=2 일때 시뮬레이션에 사용된 동작 테스트 벡터들 중에서 X=d'10, Y=d'13인 경우에 대한 모의 동작 실험결과로써 알고리즘에 따른 연산 결과값 (H'0000000000000002)과 모의 동작 실험결과가 동일함을 확인하였다.

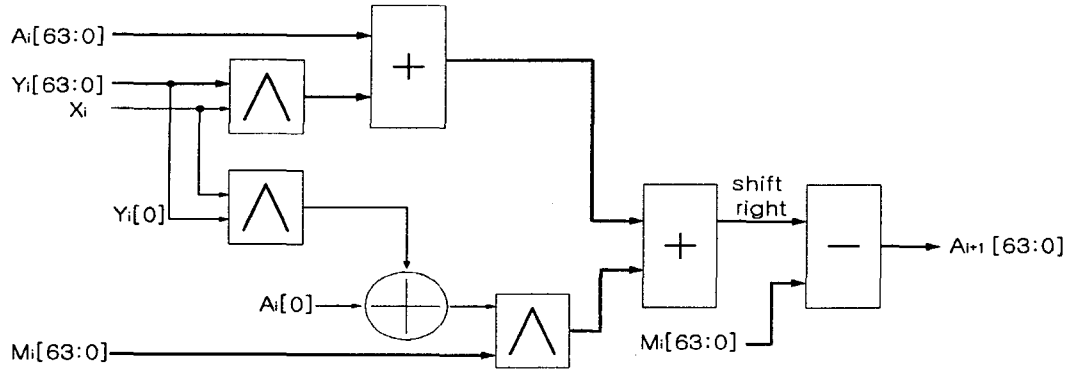

그림 3. 64 비트 Process element
Fig. 3. 64 bit Process element

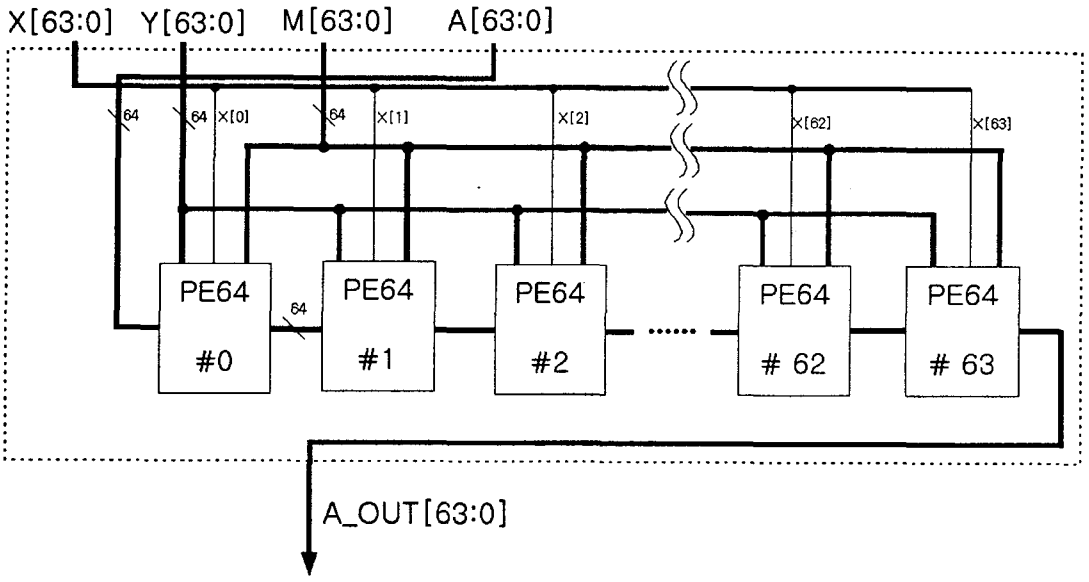

그림 4. 64 비트 모듈러 곱셈기 slice 구조
Fig. 4. 64 bit Modular multiplier's slice architecture

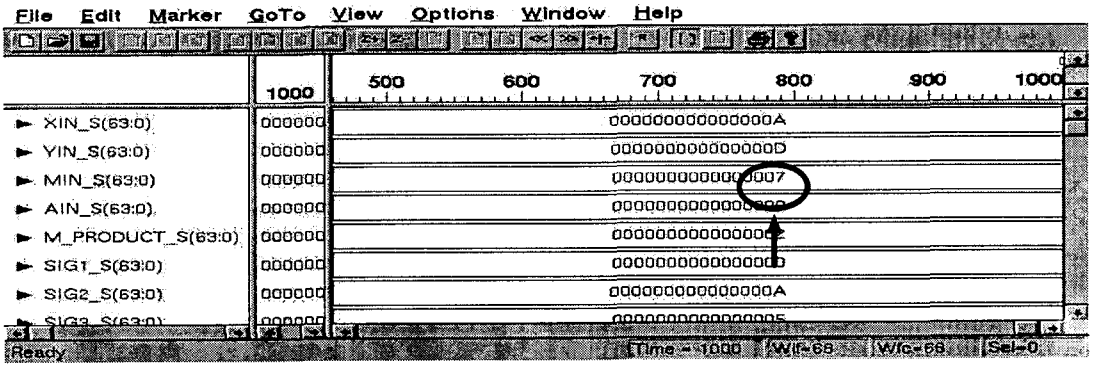

그림 5. 64 비트 bit-slice 모듈러 곱셈기의 동작파형
Fig. 5. 64 bit bit-sliced Modular multiplier's simulation result

3.2 모듈러 곱셈기의 비트 확장

제안한 알고리즘은 오퍼랜드의 비트 확장을 고려하여 bit-slice 구조로 되어 있다. slice의 크기가 n일 경우 4개의 bit-slice형 모듈러 곱셈기 slice가 조합되어 2n 비트의 모듈러 곱셈을 수행

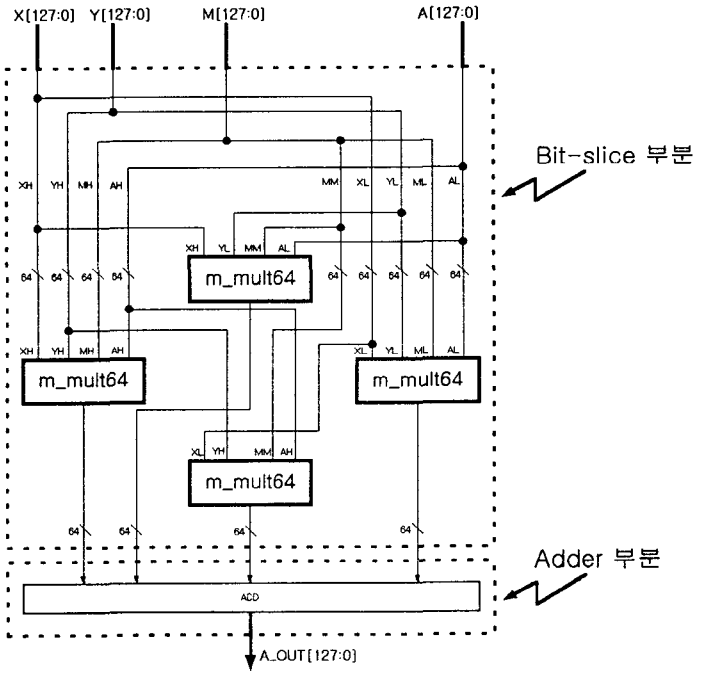

그림 6. 128 비트 모듈러 곱셈기의 내부연결
Fig. 6. Interconnection of 128 bit modular multiplier

할 수 있다. 2n 비트의 입력 X, Y를 각각 n 비트 크기인 상위 비트군(X_H, Y_H)과 하위 비트군(X_L, Y_L)을 구분하여 X_H, Y_H 를 입력으로 하는 n 비트 모듈러 곱셈기와, X_H, Y_L 를 입력으로 하는 n 비트 모듈러 곱셈기와, X_L, Y_H 를 입력으로 하는 n 비트 모듈러 곱셈기와, 모두 하위 비트군(X_L, Y_L)인 n 비트 모듈러 곱셈기를 사용함으로써 2n 비트 오퍼랜드에 대한 모듈러 곱셈기로 확장할 수 있다. 그림 6에서 128 비트로 확장되기 위해 필요한 내부 연결관계를 보인다. 이때, bit-slice 부분에서 수행된 4개의 64 비트 모듈러 곱셈 연산결과는 Adder 부분에서 함께 덧셈 연산과정이 수행된 후 비로소 확장된 비트로써 출력된다.

n 비트의 오퍼랜드를 갖는 모듈러 곱셈기를 구성하고자 할 때에는 $[\frac{n}{64}]^2$개의 64 비트 bit-slice형 모듈러 곱셈기를 이용하되 32 비트씩 분할된 입력에 대하여 각각 모듈러 연산을 수행한 후 캐리 전파 덧셈기에서의 덧셈으로 n 비트의 결과값을 출력한다.

64 비트 bit-slice형 모듈러 곱셈기로부터 128/256/512 비트 모듈러 곱셈기로 확장시키기 위해서는 각각 64 비트 bit-slice형 모듈러 곱셈기가 4개/16개/64개 사용된다. 그림 7에는 64비트 bit-slice형 모듈러 곱셈기를 이용하여 구성한 128 비트 및 256 비트 모듈러 곱셈기의 구조를 보이고 있다. 그림 7의 짙은 음영부분으로 표시된 부분이 64 비트 bit-slice형 모듈러 곱셈기를 이용한 256 비트 모듈러 곱셈기이며 하단부의 덧셈기는 각 모듈러 곱셈기의 비트 크기별로 덧셈기의 비트 크기가 결정된다.

3.3 동작검증

표준 하드웨어 기술 언어(VHDL)로부터 합성된 64 비트 bit-slice형 모듈러 곱셈기 회로를 VHDL 시뮬레이션 툴을 이용하여 모의 동작 실험한 결과(그림 8)와 FPGA칩으로 구현된 64비트 bit-slice형 모듈러 곱셈기를 로직 분석기를

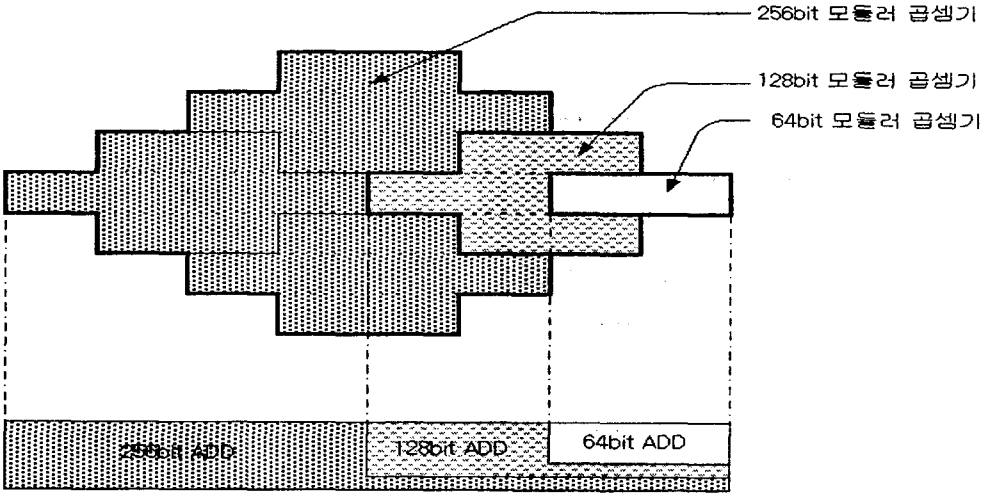

그림 7. 64 비트 Bit-sliced 모듈러 곱셈기를 이용한 128비트 및 256 비트 모듈러 곱셈기
Fig. 7. 128/256 bit modular multiplier from 64bit bit-sliced modular multiplier

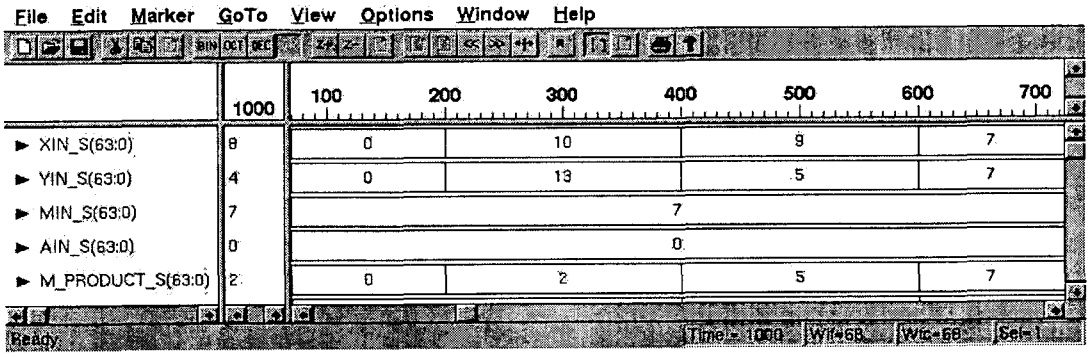

그림 8. 64 비트 bit-slice형 모듈러 곱셈기의 동작

Fig. 8. 64bit bit-sliced modular multiplier's simulation result

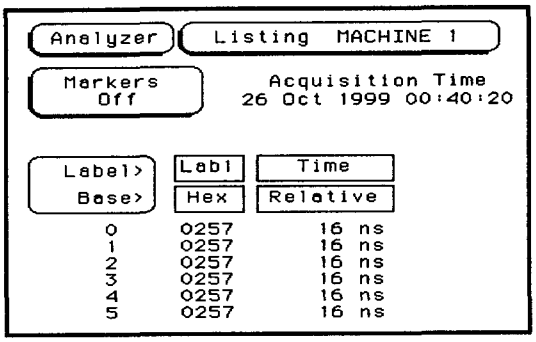

그림. 9. FPGA로 구현된 64Bit slice형 모듈러 곱셈기의 logic analyzer 분석결과

Fig. 9. FPGA implementation's functional output captureed by a logic-analyzer

이용한 동작 분석결과(그림 9)를 비교하였다.

그림 8의 동작 파형은 VHDL 시뮬레이션 툴에서 사용된 다수의 동작 테스트 벡터 중에서 임의로 선택된 4가지 동작 테스트 벡터(X1, Y1)=(D'0, D'0), (X2, Y2)=(D'10, D'13), (X3, X3)=(D'9, D'5), (X4, Y4)=(D'7, D'7)에 대한 동작 파형으로 알고리즘에 따른 수작업 계산의 결과와 동일한 결과를 보인다.

그림 9의 리스트는 FPGA칩의 출력 값을 로직 분석기의 동작 시점부터 최초 6회에 걸쳐 획득한 값이다. FPGA칩의 동작 검증에서는 FPGA칩의 팩키지 핀 수의 제한으로 동작 테스트 벡터를 내부 회로로 구현하여 연산회로에 연속적으로 인가하는 테스트 방법을 사용했기 때문에 로직 분석기의 동작 분석 결과는 6회 모두 동일한 결과를 보인다. 로직 분석기의 결과값은 FPGA칩의 출력 신호들 중에서 하위 4 비트씩 측정하여 16진 값으로 출력한 값이다. VHDL 시뮬레이션 툴에서의 결과와 로직 분석기에서의 결과는 동일함(D'0, D'2, D'5, D'7)을 확인하였다.

Ⅳ. 결론

본 논문에서는 RSA 공개키 암호시스템의 암호키(key)의 비트 크기 확장에 따른 내부 연산기의 오퍼랜드 확장에 용이하게 대응 할 수 있도록 bit-slice형 모듈러 곱셈알고리즘을 제안하였다. 제안한 알고리즘에 따라 64 비트 bit-slice

형 모듈러 곱셈기를 표준 하드웨어 기술언어(VHDL)를 이용하여 구현하였고, 이를 기본으로 하여 확장된 128/256/512 비트 모듈러 곱셈기 회로를 구현하였다. 128/256/512 비트 모듈러 곱셈기의 동작을 VHDL 시뮬레이션 툴을 이용하여 알고리즘의 동작을 검증하였으며 로직 분석기를 이용하여 FPGA칩으로 구현된 bit-slice형 모듈러 곱셈기의 동작을 확인하였다.

암호 공격에 대한 대응 방법으로 암호키(key)의 비트 크기가 확장되더라도 암호 시스템 설계자는 내부 연산기들을 재설계 하는 대신 제안된 bit-slice형 모듈러 곱셈기를 부가 장착하면 확장된 오퍼랜드의 비트 크기를 만족시킬 수 있어 새로운 암호키(key)의 비트 크기에 따른 하드웨어 재설계 문제가 쉽게 해결될 수 있다.

향후 연구방향으로는 PE를 최적화 시켜 모듈러 곱셈기의 지연시간과 회로크기를 줄이는 연구와 더불어 각 PE들을 병렬처리 하여 연산속도를 향상시키는 연구가 필요하다.

참고문헌

[1] 이석래 외, "모듈러 지수승 연산 알고리듬," 한국통신보호학회지, 제2권, 제3호, pp.90-100, 1992. 9

[2] 한국전자통신연구원, 현대 암호학, pp.128-131, 1991.

[3] 박태규, 황대준, "고속 암호화 기법에 대한 분석," 한국통신보호학회지, 제3권 제2호 pp.85-95, 1993. 6.

[4] 홍성민, 오상엽, 윤현수, "모듈라 멱승 연산의 빠른 수행을 위한 덧셈사슬 휴리스틱과 모듈라 곱셈 알고리즘들," 한국통신정보보호학회논문지, 제7권, 제2호, pp.74-91, 1997. 6.

[5] P. L. Montgomery, "Modular multiplication without trial division," Mathematics of Computation, Vol.44, No.170, pp.519-521, April 1985.

[6] A. J. Menezes, P. C. van Oorschot and S. A. Vanstone. Applied Cryptography. 4th ed., CRC Press, 1999.

[7] H. S. Hwang, et al., "An implementation and analysis of modular multiplication on PC," KIISC Review, Vol.4, No.3, pp. 34-60, 1994.

[8] C. D. Walter, "Systolic modular multiplication," IEEE Trans. on Computers, Vol.42, No.3, pp.376-378, 1993.

[9] C. K. Koc, T. Acar and B. S. Kaliski Jr., "Analyzing and comparing Montgomery multiplication algorithms," IEEE Micro, Vol.16, No.3, pp.26-33, June 1996.

[10] C. K. Koc, "Montgomery reduction with even modulus," IEE Proceedings: Computers and Digital Techniques, Vol.141, No.5, pp.314-316, September 1994.

[11] P. Kornerup, "High-radix modular multiplication for cryptosystems," IEEE Symposium on Computer Arithmetic, July 1993.

[12] F. Tenca and C. K. Koc, "A scalable architecture for montgomery multiplication. cryptographic hardware and embedded systems," CHES 99, Lecture Notes in Computer Science, No.1717, pp. 94-108, 1999.

[13] 서광석 외, 수론과 암호학, 京文社, 1998.

[14] C. K. Koc and T. Acar, "Montgomery multiplication in GF(2^k)," Proceedings of Third Annual Workshop on Selected Areas in Cryptography, Vol.14, No.1, pp.95-106, August, 1996.

Bit-slice Modular multiplication algorithm

Dhongyeol, Ryu*/Kyongrok, Cho**/Younggap, You***

Abstract

In this paper, we propose a bit-sliced modular multiplication algorithm and a bit-sliced modular multiplier design meeting the increasing crypto-key size for RSA public key cryptosystem. The proposed bit-sliced modular multiplication algorithm was designed by modifying the Walter's algorithm. The bit-sliced modular multiplier is easy to expand to process large size operands, and can be immediately applied to RSA public key cryptosystem.

* Dept. of Computer and Communication Eng. Chung-Buk National University
** Dept. of Computer and Communication Eng. Chung-Buk National University
*** Dept. of Computer and Communication Eng. Chung-Buk National University