情報保護學會論文誌
第17卷 第6號, 2007. 12

래딕스-4 몽고메리 곱셈기 기반의 고속 RSA 연산기 설계

구 본 석[1†], 유 권 호[1], 장 태 주[1], 이 상 진[2‡]
[1]한국전자통신연구원 부설연구소, [2]고려대학교 정보경영공학전문대학원

Design of high-speed RSA processor based on radix-4 Montgomery multiplier

Bonseok Koo[1†], Gwonho Ryu[1], Taejoo Chang[1], Sangjin Lee[2‡]
[1]Attached Institute of ETRI,
[2]Graduate School of Information Management & Security, Korea University

요 약

본 논문에서는 래딕스-4 몽고메리 곱셈기 기반의 고속 RSA 연산기를 제안하고 그 구현 결과를 제시한다. 캐리저장 가산기 기반의 래딕스-4 몽고메리 곱셈기를 제안하고, 중국인의 나머지 정리를 적용할 수 있도록 그 구조를 확장하였다. 이를 바탕으로 설계한 1024-비트 RSA 연산기는 1024-비트 모듈러 지수승을 0.84M 클락 사이클, 512-비트 지수승은 0.25M 클락 사이클 동안 각각 계산할 수 있으며, 0.18um 공정을 이용하여 구현한 결과, 최대 300MHz 클락 속도를 가지므로 1024-비트 지수승은 365Kbps, 512-비트 지수승은 1,233Kbps의 성능을 각각 가진다. 또한 고속 RSA 암호 시스템의 구현을 위해, 몽고메리 매핑 계수 계산 및 중국인 나머지 정리의 전처리 과정에 적용할 수 있도록 모듈러 감산 기능을 하드웨어로 구현하였다.

ABSTRACT

RSA is one of the most popular public-key crypto-system in various applications. This paper addresses a high-speed RSA crypto-processor with modified radix-4 modular multiplication algorithm and Chinese Remainder Theorem(CRT) using Carry Save Adder(CSA). Our design takes 0.84M clock cycles for a 1024-bit modular exponentiation and 0.25M cycles for a 512-bit exponentiations. With 0.18um standard cell library, the processor achieves 365Kbps for a 1024-bit exponentiation and 1,233Kbps for two 512-bit exponentiations at a 300MHz clock rate.

Keywords : *RSA processor, High-speed, High-radix, Chinese Remainder Theorem*

Ⅰ. 서 론

RSA [1] 는 현재 가장 널리 사용되고 있는 공개키 암호시스템 중 하나이다. RSA 연산은 모듈러 지수승을 기반으로 하며 그 계산 과정은 반복적인 모듈러 곱셈으로 이루어진다. 따라서 모듈러 곱셈은 RSA 암호시스템의 전체 성능을 좌우하는 중요한 요소이다. 현재까지 많은 모듈러 곱셈 알고리즘들이 제안되어 왔으며 [2-4], 몽고메리 곱셈 알고리즘 [4-5] 은 가장 널리 사용되고 있는

접수일: 2007년 6월 8일; 채택일: 2007년 9월 19일
† 주저자, bskoo@ensec.re.kr
‡ 교신저자, sangjin@korea.ac.kr

알고리즘 중 하나이다. 몽고메리 곱셈 알고리즘은 모듈러스에 대한 나눗셈 대신에 두 번의 덧셈과 쉬프트를 반복하여 모듈러 곱셈을 수행하는 방법이다. 따라서 간단한 구조로 인해 하드웨어 구현에 적합한 장점을 가진다. 암호학적인 강도를 만족시키기 위해 RSA 암호의 키 크기는 일반적으로 512~1024, 또는 2048비트가 주로 사용되고 있다. 수천 비트 값에 대한 덧셈은 캐리전파 지연시간이 매우 크기 때문에 이를 고속으로 처리하는 연산기를 구현하는 일은 간단하지 않다. RSA 연산을 보다 고속으로 구현하기 위한 하나의 방법으로 하이 래딕스(high-radix) 모듈러 곱셈 알고리즘이 제안되었다 [6-9]. 하이 래딕스 곱셈 알고리즘은 피승수의 멀티 비트를 한 클락 사이클에 처리함으로써 곱셈에 소요되는 전체 클락 사이클을 줄이는 방법이다. RSA 연산을 고속화하는 또 다른 방법은 중국인의 나머지 정리(Chinese Remainder Theorem, CRT) 기법을 사용하는 것이다 [10-12]. 중국인의 나머지 정리는 분할정복(divide-and-conquer) 방식의 기법으로서, 이 방법을 사용하면 개인키를 이용한 RSA 연산을 4배 정도 빠르게 구현할 수 있다.

일반적으로 몽고메리 모듈러 곱셈 알고리즘을 하드웨어로 구현하는 방법에는 캐리전파 지연을 피하기 위해 캐리저장 가산기(Carry Save Adder, CSA)를 이용하여 설계하는 방법 [13-15] 과 1-비트 전가산기를 기본 구조로 하는 시스톨릭 어레이(systolic- array) 구조로 설계하는 방법 [16-19] 이 있다. 캐리저장 가산기를 이용하는 방법은 입력값의 크기가 증가해도 캐리전파 지연시간이 늘어나지 않기 때문에 큰 입력값에 대한 고속 연산을 수행하는데 유리한 반면, 하이 래딕스 곱셈기를 설계하는 경우에는 캐리저장 형태의 중간 결과값으로부터 최하위 비트 주변값을 결정하기 어려운 문제가 있다. 또한 시스톨릭 어레이 구조의 설계 방법은 1-비트 전가산기를 기본구조로 파이프라이닝(pipelining) 형태를 가지므로 고속 구현에 적합하지만, 하이 래딕스 곱셈기로 설계하는 경우는 파이프라이닝 구조 때문에 대기 시간(latency)이 발생하고 컨트롤이 복잡해지는 문제가 있다.

본 논문에서는 캐리저장 가산기 기반의 래딕스-4 몽고메리 모듈러 곱셈기를 설계하고, 이를 수정하여 중국인의 나머지 정리 기법도 활용할 수 있는 고속 RSA 연산기의 설계 방법을 제시한다. 아울러 제안하는 고속 RSA 연산기에는 모듈러 지수승 뿐만 아니라, 몽고메리 매핑 계수와 중국인 나머지 정리 기법의 RSA 연산에 필요한 모듈러 감산기능도 함께 설계한다.

본 논문은 다음과 같이 구성된다. 2장에서는 제안하는 RSA 연산기에 사용된 알고리즘, 3장에서는 RSA 연산기의 하드웨어 아키텍쳐를 설명하고, 4장에서는 설계한 연산기를 CMOS ASIC 라이브러리로 합성한 결과를 제시하고 기존에 발표된 결과들과 비교하며, 마지막으로 5장에서는 결론을 맺는다.

Ⅱ. 알고리즘

2.1. 래딕스-4 몽고메리 모듈러 곱셈

모듈러 곱셈의 성능을 개선하기 위해 많은 하이 래딕스 모듈러 곱셈 알고리즘들이 제안되었는데 [6-9], 최근에는 Hong 등이 변형된 부스(Booth) 인코딩을 이용한 래딕스-4 몽고메리 곱셈 알고리즘을 제안하였다. 이렇게 변형된 부스 인코딩을 이용하면, 피승수 B와 모듈러스 N의 배수 정보로 $\{B, 2B, 3B\}$와 $\{N, 2N, 3N\}$ 대신, $\{\pm B, \pm 2B\}$와 $\{\pm N, \pm 2N\}$만을 사용하게 되며, 배수 정보 $3B$나 $3N$을 계산하기 위해서는 별도의 가산기가 필요한 반면에 배수 정보 $2B$와 $2N$은 1-비트 좌측 쉬프트로 간단히 구현되므로 하드웨어 비용을 절약할 수 있다 [9]. 하지만 이 래딕스-4 알고리즘을 캐리저장 가산기로 설계하는 경우에는, mod 4와 나누기 4 연산은 C와 S를 직접 더하기 전에는 수행할 수 없다. 따라서, 본 논문에서는 2-비트 덧셈을 이용하여, Hong 등의 래딕스-4 알고리즘을 캐리저장 가산기에 적용할 수 있게 수정한다. 3-비트 $(a_{2i+1}, a_{2i}, a_{2i-1})$과 n-비트 수 B를 입력받아, [표 1]과 같이 neg_B와 B_i를 출

[표 1]. 알고리즘 1의 변형된 래딕스-4 부스 인코딩 규칙 ($BR((a_{2i+1}, a_{2i}, a_{2i-1}), B)$)

a_{2i+1}	a_{2i}	a_{2i-1}	neg_B	Bi
0	0	0	0	0
0	0	1	0	B
0	1	0	0	B
0	1	1	0	$2B$
1	0	0	0	$\neg 2B$
1	0	1	0	$\neg B$
1	1	0	0	$\neg B$
1	1	1	0	0

력하는 변형된 래딕스-4 부스 인코딩을 $BE((a_{2i+1},a_{2i},a_{2i-1}),B)$로 표현할 때, 수정된 캐리저장 가산기 기반의 래딕스-4 모듈러 곱셈 방법은 알고리즘 1과 같다.

알고리즘 1에서 $(C,S)=CSA(X,Y,Z)$은 세 입력 X, Y, Z에 대한 캐리저장 덧셈을 나타내며, 그 결과는 각각 C와 S로 저장된다. 캐리전파 가산기(Carry Propagation Adder, CPA)와는 달리, 캐리저장 가산기는 별도의 캐리 입력이 없기 때문에, B나 N에 대한 2의 보수를 계산을 직접 계산하는 것은 불가능하다. 따라서 [표 1]에서처럼 부스 인코딩 결과가 양수를 출력하는 경우에는 neg_B는 0, B_i는 B 혹은 $2B$가 되며, 음수를 출력하는 경우에는 neg_B는 1, B_i는 B 혹은 $2B$의 비트별 역원인 $\neg B$나 $\neg 2B$가 된다. 같은 방식으로 N의 배수도 neg_N과 N_i로 표현된다.

알고리즘 1: 캐리저장 가산기 기반의 수정된 래딕스-4 부스 몽고메리 곱셈, R4MM(A,B,N)

입력: $A=\Sigma_{i=0}^{n-1}a_i2^i$, $B=\Sigma_{i=0}^{n-1}b_i2^i$, $N=\Sigma_{i=0}^{n-1}n_i2^i$

출력: $C=ABR^{-1} \bmod N$, $R=2^{\lceil (n+3)/2 \rceil} \bmod N$

1. $C=0, S=0, a_{-1}=0, c_{in}=0$
2. **for** $i=0$ **to** $\lceil (n+3)/2 \rceil -1$ **do begin**
3. $(negB, B_i)=BE((a_{2i+1},a_{2i},a_{2i-1}),B)$
4. $(C,S)=CSA(S,C,B_i)$
5. $(t_{i1},t_{i0})=((s_1,s_0)+(c_1,\neg_B)+c_{in}) \bmod 4$
6. $neg_N=(t_{i1}=n_1)$
7. **if** $(t_{i0}=0)$ **do begin**
8. **if** $(t_{i1}=0)$ $N_i=0$
9. **else** $N_i=2N$
10. **end**
11. **else do begin**
12. **if** $(neg_N=1)$ $N_i=\neg N$, $C=C+neg_N$
13. **else** $N_i=N$
14. **end**
15. $(C,S)=CSA(S,C,N_i)$
16. $c_{in}=((s_1,s_0)+(c_1,neg_B)+c_{in})/4$
17. $(C,S)=(S/4,C/4)$
18. **end**
19. $S=C+S$
20. **return** S

단계 5에서 두 개의 2-비트 입력 (s_1, s_0)와 (c_1, neg_B)와 캐리 입력 c_{in}에 대한 덧셈을 취한 후 mod 4 연산을 수행하는데, 이는 전체 덧셈 결과 중 최하위 2-비트만 결과값 (t_{i1}, t_{i0})으로 취함으로써 간단하게 처리된다. 단계 6~15에서는 (t_{i1}, t_{i0})와 (n_1, n_0)값의 조건에 따라 N_i와 neg_N을 결정하고 이를 중간 결과값 (C,S)에 더하는 과정을 수행한다. 이때 단계 12의 $C+neg_N$은 C의 최하위 비트가 항상 0이기 때문에 이를 neg_N으로 대체하면 된다. 한편 단계 16에서의 2-비트 덧셈과 나누기 4는 2-비트 덧셈 후 발생하는 캐리만을 c_{in}으로 업데이트시키는 방식으로 간단히 처리되며, 단계 17에서 나누기 4는 중간 결과값 (C,S)의 최하위 두 비트가 모두 0이므로 C와 S를 각각 오른쪽으로 2-비트 쉬프트 시키는 과정이 된다. 결과적으로 이러한 두 번의 2-비트 덧셈과 캐리 갱신 방식을 사용하여, Hong 등의 래딕스-4 부스 몽고메리 곱셈 알고리즘을 캐리저장 가산기 기반의 곱셈기로 설계할 수 있다. 한편, 이러한 알고리즘을 래딕스-8로 보다 확장할 경우, 피승수 B와 모듈러스 N의 배수 정보로 $\{\pm B, \pm 2B\}$와 $\{\pm N, \pm 2N\}$이 아닌, $\{\pm B, \pm 2B, \pm 3B\}$와 $\{\pm N, \pm 2N, \pm 3N\}$이 필요하며, $2B$나 $2N$은 1-비트 좌측 쉬프트로 간단히 구현되는 반면, $3B$나 $3N$을 계산하기 위해서는 별도의 가산기가 부가적으로 필요하여 연산기의 하드웨어 복잡도 및 지연시간을 가중시키므로, 오히려 전체적인 성능이 저하되는 결과를 가져올 수 있다.

2.2. 모듈러 지수승과 중국인의 나머지 정리

앞서 언급한 바와 같이 모듈러 지수승은 모듈러 곱셈의 반복 수행하여 계산할 수 있다. 알고리즘 2는 몽고메리 곱셈을 이용한 이진 몽고메리 지수승 알고리즘이다. 여기서 $R4MM(\ )$은 알고리즘 1의 래딕스-4 몽고메리 곱셈 알고리즘을 나타낸다.

단계 1은 초기값 M과 1을 매핑 계수 K와 각각 몽고메리 곱셈을 수행함으로써 몽고메리 도메인으로 매핑하는 과정이다. 여기서 매핑 계수 K는 모듈러스 N이 주어지면 결정되는 상수값이며, 모듈러 지수승 연산을 수행하기 전에 미리 계산되어야 한다. 단계 2~5는 지수 e값의 조건에 몽고메리 곱셈을 반복하여 모듈러 지수승 연산을 수행하는 주요 과정이며, 단계 6에서는 몽고메리 도메인에서의 지수승 연산 결과를 다시 정수 도메인으로 매핑시키는 과정이다. 한편 단계 7에서는 결과값

알고리즘 2: 이진 몽고메리 지수승 알고리즘

입력:

$M=\Sigma_{i=0}^{n-1} a_i 2^i$, $N=\Sigma_{i=0}^{n-1} n_i 2^i$, $e=\Sigma_{i=0}^{n-1} e_i 2^i$,
$m_i, e_i, n_i \in 0,1$, and $K=R^2 \bmod N = 4^{\lceil (n+3)/2 \rceil} \bmod N$

출력: $M^e \bmod N$

1. $M = R4MM(M, K, N)$, $P = R4MM(K, 1, N)$
2. **for** i = n-1 **to** 0 **do begin**
3. $P = R4MM(P, P, N)$
4. if $(e_i = 1)$ $P = R4MM(P, M, N)$
5. **end**
6. $P = R4MM(P, 1, N)$
7. **if** $(P < 0)$ $P = P + N$
8. **return** P

P의 부호를 조사하여 음수인 경우, N을 한번 더해서 최종 결과값을 얻는데, 이는 알고리즘 1의 *R4MM*( ) 곱셈의 결과가 $[N, -N)$의 범위에 해당하는 값을 출력하기 때문이다.

개인키를 이용한 RSA 연산은 중국인의 나머지 정리를 이용하여 보다 고속으로 구현할 수 있다[11-12]. 모듈러스 N은 두 개의 각기 다른 큰 소수 P와 Q의 곱으로 이루어진 수이므로, 중국인의 나머지 정리를 적용하면, 모듈러 지수승 $C^d \bmod N$은 다음과 같이 계산될 수 있다.

알고리즘 3에서 초기에 단계 1~2는 n-비트 입력값 C를 절반 크기인 $n/2$-비트 값 C_P와 C_Q로 감소시키는 과정이며, 단계 3~4는 $n/2$-비트 값 C_P와 C_Q 각각에 대해 모듈러 지수승을 수행하는 과정이며, 단계 5~6은 두 개의 $n/2$-비트 모듈러 지수승의 결과값 M_P와 M_Q을 재조합하여 최종 결과값 M을 얻는 과정이다. 여기서 다른 단계에 비해 계산 시간이 월등히 많이 소요되는 단계 3~4의 두 $n/2$-비트 모듈러 지수승은 서로 독립적인 연산이므로 병렬로 계산할 경우, 일반적인 n-비트 모듈러 지수승 방법보다 계산 속도를 4배 정도 빠르게 할 수 있다. 결론적으로 중국인 나머지 정리를 이용한 속도 향상 효과는 모듈러스와 지수의 크기를 절반으로 줄여서 지수승을 수행하는데서 기인한다.

2.3. 모듈러 지수승과 중국인의 나머지 정리

알고리즘 2에서 보았듯이 몽고메리 곱셈기를 이용한 RSA 연산에는 매핑 계수 $K=4^{\lceil (n+3)/2 \rceil} \bmod N$

알고리즘 3: 중국인 나머지 정리를 적용한 RSA 연산

입력: $C, P, Q, Q, ^{-1}$,
$d_p = d \bmod (P-1)$, $d_Q = d \bmod (Q-1)$

출력: $M = C^d \bmod N$

1. $C_P = C \bmod P$
2. $C_Q = C \bmod Q$
3. $M_P = C_P^{d_P} \bmod P$
4. $M_Q = C_Q^{d_Q} \bmod Q$
5. $h = Q^{-1}(M_P - M_Q) \bmod P$
6. $M = M_Q + hQ$
7. **return** M

알고리즘 4: 임의의 n-비트 N에 대해 부호추정 기법을 이용한 모듈러 감산 알고리즘

입력: $X=\Sigma_{i=0}^{k+n-1} x_i 2^i$, $N=\Sigma_{i=0}^{n-1} n_i 2^i$

출력: $X \bmod N$

1. $S = 2^{-k-1}X$, $C = 0$, $l = 0$
2. **while** $(N_{n-1} = 0)$ **do begin**
3. $N = 2N$, $l = l + 1$
4. **end**
5. **for** i =0 **to** $k + l$ **do begin**
6. **if** $ES(C,S) = (+)$ $(C,S) = CSA(2S, 2C, -N)$
7. **elseif** $ES(C,S) = (-)$ $(C,S) = CSA(2S, 2C, N)$
8. **else** $(C, S) = CSA(2S, 2C, 0)$
9. **end**
10. $S = C + S$
11. **if** $(S < 0)$ $S = S + N$
12. **while** $(l \neq 0)$ **do begin**
13. $S = S/2$, $l = l - 1$
14. **end**
15. **return** S

을 미리 계산하여야 하며, 중국인의 나머지 정리를 사용하여 RSA 연산을 수행할 경우에는 알고리즘 4에서처럼 $C_P = C \bmod P$나 C_Q를 먼저 계산해야 한다. 매핑 계수 K는 모든 공개키 N에 대해 미리 계산하여 저장했다가 사용할 수 있지만 메모리 공간을 소모하게 된다. 뿐만 아니라, C_P와 C_Q는 암호문 C를 상대방으로부터 전달받기 전에는 계산될 수 없는 값이다. 따라서 하드웨어 기반의 고속 RSA 암호 시스템에는 모듈러 감산 기능을 하드웨어로 구현하는 것이 바람직하다.

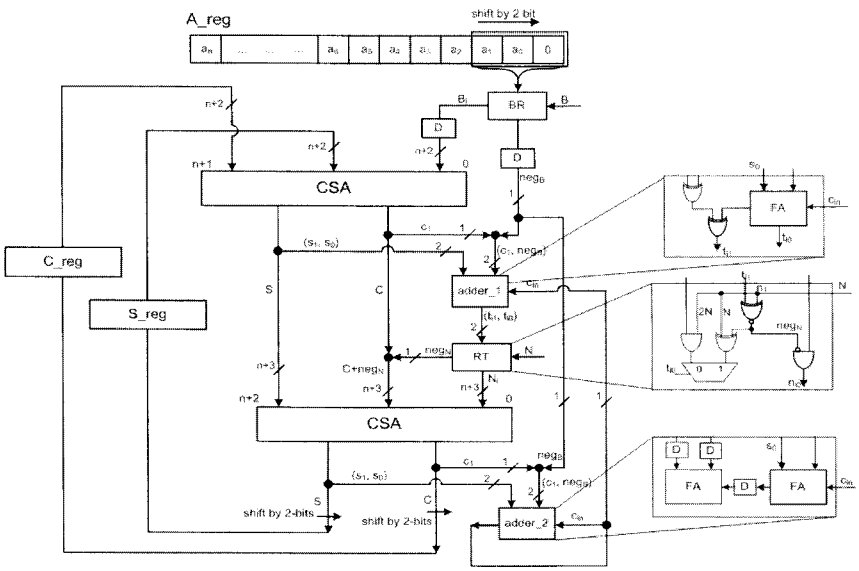
캐리저장 가산기를 이용한 방법에는 Koc 등이 제안한 부호추정 기법(sign estimation)을 이용한 모듈러 감산 알고리즘이 있다[21]. Koc 등의 모듈러 감산 알고리즘은 4-비트 덧셈을 사용하여 캐리저장 형태의 중간 결과의 부호를 추정함으로써 모든 연산을 캐리저장 가산기로 수행하도록 한다. 따라서 고속 하드웨어 구현에 매우 적합하다. 한편 Koc의 부호추정 기법을 적용하기 위해서는 모듈러스 N의 크기를 알고, 이를 표현하는데 정확히 필요한 비트 수 중에 최상위 비트를 기준으로 4-비트 덧셈을 적용하여야 한다. 하지만, RSA 암호 시스템에서 사용하는 N은 n-비트로 표현할 수 있는 임의의 값이 사용될 수 있다. 따라서, 알고리즘 4에서처럼 N의 최상위 비트가 1이 아닐 경우는 쉬프트-업 시켜서 모듈러 감산을 수행한 다음, 쉬프트-다운 시켜서 최종 결과를 계산하는 방식으로 Koc 등의 모듈러 감산 알고리즘을 적용할 수 있다.

알고리즘 4에서 $ES(C, S)$는 Koc의 모듈러 감산 알고리즘의 핵심인 부호추청 기법을 의미한다. 부호추정 기법에 대한 보다 자세한 내용은 관련 논문[21]을 참조하기 바란다. 임의의 n-비트 N에 대해 동작하도록 단계 2~4에서 N의 n-1번째 비트가 1이 될 때까지 N을 왼쪽 쉬프트하면서 t을 1씩 증가시키고, 단계 12~14에서 t이 0이 될 때까지 1씩 감소시키면서 S를 오른쪽 쉬프트시키는 과정이 원래 Koc 등의 모듈러 감산 알고리즘에 부가적으로 추가되었다.

Ⅲ. 하드웨어 아키텍처

3.1. 모듈러 곱셈기

캐리저장 가산기와 변형된 부스 인코딩을 이용하는 알고리즘 1을 기반으로 래딕스-4 몽고메리 모듈러 곱셈기를 설계하였으며, 그 구조는 그림 1과 같다. 설계된 곱셈기는 (n+2)-비트 캐리저장 가산기와 (n+3)-비트 캐리저장 가산기, 두 개의 2-비트 가산기, BE(Booth Encoding) 블록, RT(Reduction Table) 블록, 그리고 세 개의 레지스터(A_reg, B_reg, C_reg)를 주요 구성 요소로 가진다. 입력 A이 초기에 A_reg에 저장된 다음, 매 클락 사이클마다 2-비트씩 오른쪽으로 쉬프트하면서, 모듈러 곱셈을 수행하게 된다. BR 블록은 $(a_{2i+1}, a_{2i}, a_{2i-1})$와 B를 입력 받아 neg_B와 B_i를 발생시키고, RT 블록은 (t_{i1}, t_{i0})와 N으로부터 neg_N과 N_i을 생성한다. 두 입력 A와 B는 2의 보수로 표현되는 수이기 때문에 (n+1)-비트의 크기를 가지고, 모듈러스 입력 N은 n-비트 수이다. BR 블록에서 생성되는 B_i는 $\{-2B, -B, 0, B, 2B\}$ 중 하나의 값을 가진다. 따라서 위쪽에 위치한 캐리저장 가산기는 (n+2)-비트의 크기를 가지며, 아래쪽의 캐리저장 가산기는 1-비트 확장된 (n+3)-비트의 크기가 된다. 또한 2-비트 가산기 adder_1은 RT 블록의 입력값으로 사용되



〔그림 1〕 본 논문에서 제안하는 래딕스-4 모듈러 곱셈기의 기본 하드웨어 아키텍처

〔표 2〕. 캐리저장 가산기 기반 몽고메리 곱셈기의 최대지연 시간과 클락 사이클 수의 비교

	최대지연 시간	클락 사이클 수	n-bit CSA의 수
Kwon 등 [13]	2FAs +2ANDs	n+32(w=32)	2
McIvor 등 [23]	3FAs +2XORs +1AND	n+1	3
Ours (그림 1)	4FAs +2XORs +1AND	$\lceil (n+5)/2 \rceil + \lceil (n+2)/w \rceil$	2

는 (t_{i1}, t_{i0})를 계산하고, adder_2는 다음번 반복 연산에서 사용될 c_{in}을 계산한다.

제안하는 연산기는 $\lceil (n+3)/2 \rceil + 1$ 클락 사이클 후에, 캐리저장 형태의 결과가 C_reg와 S_reg에 각각 저장되며, 최종 결과값을 얻기 위해서는 w-비트 가산기를 이용하여 C_reg와 S_reg의 값을 w-비트 단위로 더하여 최종 결과값을 얻는다. 따라서 제안하는 래딕스-4 모듈러 곱셈기는 $\lceil (n+3)/2 \rceil + \lceil (n+2)/w \rceil + 1$ 클락 사이클 동안 한 번의 곱셈을 수행한다. [그림 1]에서 보듯이, 본 연산기의 최대 지연 경로는 "CSA→adder_1→RT→ CSA→adder_2"가 되며, 이 경로의 구성 로직은 "4FAs+2XORs +AND"가 되므로 수백 MHz 이상의 클락 속도를 구현할 수 있다. [표 2]에는 기존의 캐리저장 가산기 기반의 몽고메리 곱셈기와 본 논문에서 제안하는 곱셈기의 최대지연 시간과 클락 사이클 수를 비교하였다. 제안하는 곱셈기의 최대지연 시간이 다른 곱셈기에 비해 2FAs 정도 더 길지만, 클락 사이클 수가 거의 절반이 되어 고속 구현에 유리한 장점을 가짐을 알 수 있다.

앞서 설명한 바와 같이, 중국인의 나머지 정리를 이용하는 RSA 연산 과정에서는 두 개의 $n/2$-비트 모듈러 지수승이 가장 계산 시간을 많이 소요하는 핵심 연산이며, 이러한 두 $n/2$-비트 모듈러 지수승을 병렬로 처리하면 n-비트 지수승보다 약 4배 빠른 RSA 계산 속도를 얻을 수 있다. 또한 이것이 가능하기 위해서는 두 개의 $n/2$-비트 모듈러 곱셈기가 필요하다. 제안하는 모듈러 곱셈기의 중심인 캐리저장 가산기는 인접 비트들 간에 캐리전파가 발생하지 않으므로, $n/2$-비트 가산기를 절반씩만 사용하면 두 개의 $n/2$-비트 가산기를 손쉽게 구현할 수 있다. 따라서 최종적으로 중국인의 나머지 정리를 적용할 수 있도록, 하나의 n-비트 모듈러 곱셈기와 두 개의 $n/2$-비트 모듈러 곱셈기로 선택적으로 동작할 수 있는 래딕스-4 모듈러 곱셈기를 설계하였다. 설계된 곱셈기는 입력 A(또는 B)를 하나의 $(n+2)$-비트 수, 또는 두 개의 $(n/2+1)$-비트 수가 연접한 형태로 선택적으로 처리한다. 또한 N도 두 개의 $n/2$-비트 소수 P와 Q를 연접한 입력으로 처리할 수 있다. 결과적으로 제안하는 곱셈기는 n-비트 모듈러 곱셈 계산에는 $\lceil (n+3)/2 \rceil + \lceil (n+2)/w \rceil + 1$ 클락 사이클을, 두 개의 $n/2$-비트 곱셈 계산에는 $\lceil (n/2+3)/2 \rceil + \lceil (n/2+2)/w \rceil + 1$클락 사이클을 각각 소요한다.

3.2. 모듈러 감산기

RSA 연산에 부가적으로 필요한 모듈러 감산을 처리할 수 있도록, 알고리즘 4를 기반으로 고속 모듈러 감산 하드웨어를 설계하였으며, 그 구조를 [그림 2]에 나타내었다. 제안하는 모듈러 감산기는 임의의 $(n+k)$-비트 X와 n-비트 N 입력에 대해 X mod N 연산을 수행할 수 있으며, [그림 2]와 같이 $(n+2)$-비트 캐리저장 가산기, $\lceil \log_2 n \rceil$-비트 업/다운 카운터와 4-비트 가산기, 그리고 4개의 저장용 레지스터 N_reg, C_reg, S_reg, L_reg 등으로 구성된다.

제안하는 모듈러 감산기는 연산을 시작하면 N 값을 저장하는 N_reg 레지스터의 최상위 비트가 1이 될 때까지 좌측 쉬프트 시키면서, 동시에 L_reg에 저장된 $\lceil \log_2 n \rceil$-비트 수를 1씩 증가시키다. S_reg와 C_reg 레지스터는 초기에 $2^{k-1}X$와 0을 각각 저장하고, $(k+l+1)$번의 반복 연산 동안에는 캐리저장 형태의 중간 결과값을 저장한다. 앞서의 모듈러 곱셈기와 마찬가지로 w-비트 가산기를 사용하여 C_reg와 S_reg 레지스터에 저장된 값을 더해서 C_reg에 저장하며, 이 값의 최상위 비트가 1인 경우에는 N_reg 레지스터의 값을 더해준다. 마지막 과정으로 L_reg 레지스터에 저장된 값만큼 C_reg 레지스터의 값을 우측 쉬프트 시켜서 최종 결과값을 계산한다. 한편, 그림에서 4-비트 가산기는 부호추정 기법에 사용하기 위해 C_reg와 S_reg 레지스터의 상위 4-비트에 대해 덧셈을 수행하는데 사용된다. 결과적으로 제안하는 모듈러 감산기는 X가 $(n+k)$-비트 수이고 N이 상위 l이 모두 0인 n-비트 수인 경우, 한 번의 모듈러 감산을 수행하는데

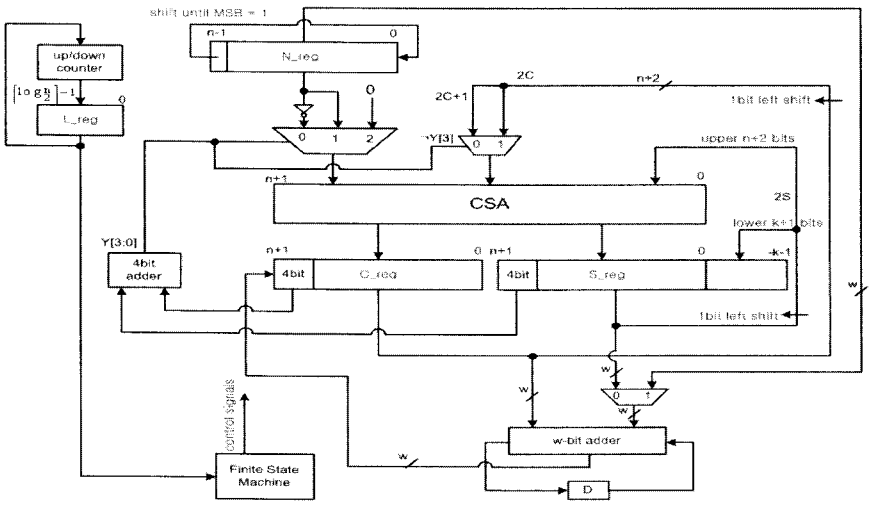


그림 2 모듈러 감산기의 하드웨어 아키텍처

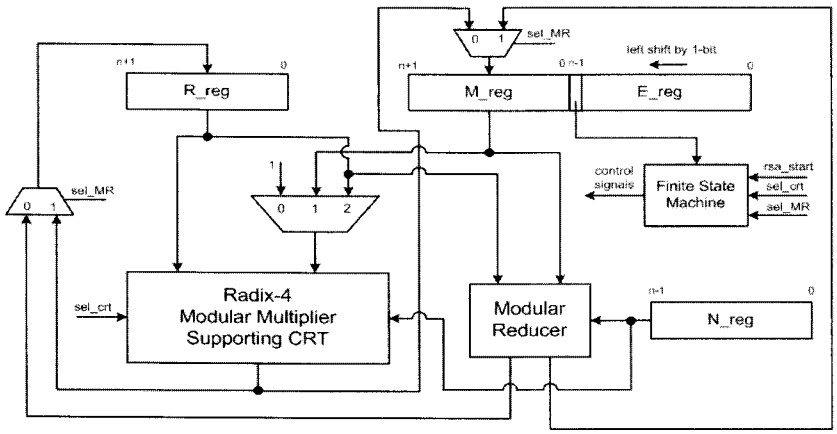


그림 3 제안하는 고속 RSA 연산기의 내부 블록도

$2l+(k+l+1)+2\lceil (n+2)/w \rceil$ 클락 사이클을 소요한다.

3.3. 고속 RSA 연산기

[그림 3]에는 최종 설계한 고속 RSA 암호 연산기의 대략적인 구조를 나타내었다. 제안하는 연산기는 앞서 설명한 래딕스-4 모듈러 곱셈기와 모듈러 감산기를 주요 구성 요소로 가지며, 모듈러 지수승이나 모듈러 감산을 선택적으로 수행할 수 있다.

입력 신호 sel_MR이 0인 경우, 본 연산기는 알고리즘 2의 이진 몽고메리 지수승 방법으로 모듈러 지수승 $M^e \bmod N$ 연산을 수행한다. 이 경우, N_reg와 M_reg, E_reg 레지스터는 각각 *N, M, e* 값을 저장하며, 최종 결과값은 R_reg 레지스터에 저장된다. 여기서 사용된 래딕스-4 몽고메리 곱셈기는 앞서 설명한 것처럼 중국인의 나머지 정리를 적용할 수 있으므로, 입력 신호 sel_crt가 0인 경우 *n*-비트 지수승을 수행하고, sel_crt가 1인 경우는 두 개의 *n*/2-비트 모듈러 지수승을 병렬로 수행한다. 또한 이진 몽고메리 지수승을 사용하므로, *n*-비트 지수승 연산에는 평균 $1.5(n^2/2+n^2/w)$ 클락 사이클이, 두 개의 *n*/2-비트 모듈러 지수승에는 평균 $1.5(n^2/8+n^2/4w)$클락 사이클이 각각 소요된다.

한편, 입력 신호 sel_MR이 1인 경우는 모듈러 감산을 수행하는데 이 경우 별도의 레지스터 공간을 두지 않고 하드웨어 자원 절약을 위해 M_reg와 E_reg 레지스터가 X 값을 저장한다. 따라서 X는 최대 $(2n+2)$-비트의 값이 될 수 있다. 또한 모듈러 감산 결과는 R_reg 레지스터에 저장된다. 본 연산기는 N의 최상위 비트가 1인 경우, 몽고메리 매핑 계수 K를 계산하는 데는 $n+3+2\lceil (n+2)/w \rceil$ 클락 사이클이 소요되고, P(또는 Q)의 최상위 비트가 1인 경우, 모듈러 감산된 암호문 C_P (또는 C_Q) 계산하는 데는 $1.5n+3+2\lceil (n+2)/w \rceil$ 클락 사이클이 소요된다.

Ⅲ. 구현 결과

이제까지 설명한 내용으로 1024-비트 고속 RSA 암호 연산기를 Verilog HDL로 설계하고 삼성 0.18um CMOS 공정으로 합성한 결과, 최대 약 300MHz의 동작 클락 속도를 보였다. 해당 연산기는 고속 구현을 위해 32-비트 (w=32) 캐리 룩어헤드 가산기를 사용하였으며, 1024-비트 지수승 한 번에 약 0.84M 클락 사이클을, 512-비트 지수승 두 번에 약 0.25M 클락 사이클을 사용한다. 따라서 300MHz 클락 속도를 기준으로 1024-비트 지수승의 경우 365Kbps, 512-비트 지수승의 경우 1,233Kbps의 속도를 가진다. [그림 4]에는 설계한 1024-비트 고속 RSA 연산기의 합성 결과 로그를 나타내었으며, [표 3]에는 연산기의 내부 구성 요소별로 하드웨어 복잡도를 정리하여 나타내었다. 여기서

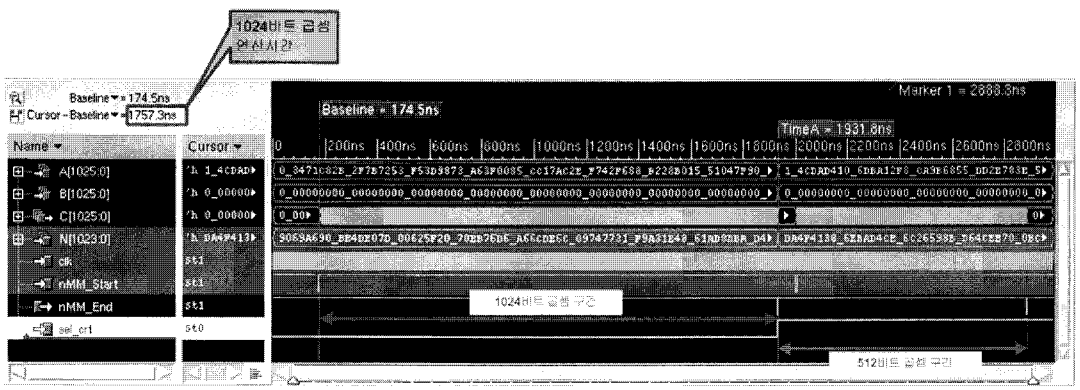
```
1
2 ****************************************
3 Report : reference
4 Design : RSA1024_HS
5 Version: W-2004.12-SP2-1
6 Date   : Mon Sep 17 13:56:26 2007
7 ****************************************
8
9 Reference          Library      Unit Area   Count    Total Area   Attributes
10 --------------------------------------------------------------------------
11 R4MM                          86583.000000     1  86583.000000  h, n
12 MR                            32861.000000     1  32861.000000  h, n
13 R_REG                          6840.000000     1   6840.000000  h, n
14 M_REG                          6840.000000     1   6840.000000  h, n
15 E_REG                          6827.000000     1   6827.000000  h, n
16 N_REG                          6827.000000     1   6827.000000  h, n
17 RSA1024_DW01_add_33_1           535.333435     1    535.333435  h
18 ......          std130         1.666670     4      6.666680
19 ......          std130         2.333330     4      9.333320
20 --------------------------------------------------------------------------
21 Total 123 references                                189455.343750
```

〔그림 4〕 제안하는 고속 RSA 연산기의 합성 결과 로그

〔표 3〕 제안하는 1024-비트 RSA 연산기의 하드웨어 복잡도

구성 요소	개수	게이트 카운트
R4MM	1	86,583
MR	1	32,861
R_REG	1	6,840
M_REG	1	6,840
E_REG	1	6,827
N_REG	1	6,827
Controller	1	42,677
Total		189,455

R4MM은래딕스-4 몽고메리 곱셈기(그림 1)를 나타내며, MR은 모듈러 감산기(그림 2), 그리고 R_reg, M_reg, E_reg, N_reg는 [그림 3]에 도시된 각 레지스터들을 나타낸다.



〔그림 5〕 래딕스-4 1024-비트 모듈러 곱셈기의 시뮬레이션 출력 파형 (300MHz 클락 속도 기준)

[그림 5]에는 제안하는 래딕스-4 1024-비트 모듈러 곱셈기에 대해, $C = A \times B \bmod N$ 연산 동작을 시뮬레이션한 결과 파형을 나타내었다. 그림에서 nMM_Start 신호는 곱셈기의 동작 시작을, nMM_End 신호는 완료를 각각 나타내는 신호이며, sel_crt 신호는 1024-비트 및 512-비트 동작을 선택하는 신호이다. 그림에서 1024-비트 모듈러 곱셈 시간을 살펴보면, 1,757ns ≃ 3.33ns(클락 주기) x 1024(비트 크기) / 2 (래딕스-4)의 연산 시간을 가진다. 아울러, 512-비트 곱셈의 경우는 1024-비트 연산 시간의 절반 정도의 시간이 소요된다.

[표 4]는 기존에 발표된 캐리저장 가산기 기반의 RSA 연산기들과 제안하는 연산기의 성능을 비교하여 나타내었다. 표에서 보듯이 본 연산기는 발표된 연산기들 중 가장 높은 성능을 보인다. Kwon 등 [13] 은 캐리저장 가산기를 이용한 기본적인 래딕스-2 모듈러 곱셈기를 사용하였으므로, 성능이 그다지 높지 않다. Blum 등 [6] 은 클락 사이클 수를 줄이기 위해 래딕스-16 모듈러 곱셈기를 FPGA를 타겟으로 설계하였으나, FPGA 내부에 탑재된 메모리 자원을 이용하여 하이 래딕스 계산에 필요한 값들을 미리 계산하는 방식을 사용하였다. 한편 Cho 등 [22] 은 캐리저장 가산기와 부호 추정 방법을 이용한 래딕스-4 모듈러 곱셈기를 설계하였다. 하지만 여기서는 n-비트 캐리저장 가산기 4개를 사용하여 설계하였으므로, 상대적으로 많은 하드웨어 자원을 소모한다. 마지막으로 McIvor [23] 가 설계한 모듈러 곱셈기는 3개의 n-비트 캐리저장 가산기를 이용하여 모든 중간 결과값을 캐리저장 형태로 저장하는 방식으로, w-비트 단위의 덧셈 과정이 필요 없는 장점을 가진다. 하지만 이러한 구조 또한 많은 하드웨어 자원을 소모하는데, 예를 들어 1024-비트 곱셈기를 구현하는데 165K 게이트 카운트를 소모한다.

한편 제안하는 연산기는 중국인의 나머지 정리를 선택적으로 적용할 수 있는 반면에, 기존 연산기들은 이를 고려하지 않거나, 중국인의 나머지 정리를 사용한 연산만이 가능하도록 설계되었다. 더불어서 본 연산기는 RSA 연산 과정에서 필요한 모듈러 감산을 위해 2050-비트 수 X와 1024-비트 수 N에 대해 $X \bmod N$연산을 선택적으로 수행할 수 있다. 따라서 본 논문에서 제안하는 RSA 연산기는 하드웨어 기반의 고속 RSA 암호 시스템을 구현하는데 매우 효과적이라고 할 수 있다.

Ⅳ. 결 론

본 논문에서는 캐리저장 가산기를 기반으로 고속의 RSA 연산기를 설계하고 구현하는 방법을 제안하였다. 제안하는 연산기는 래딕스-4 모듈러 곱셈을 사용하며, 중국인의 나머지 정리를 적용할 수 있다. 1024-비트 RSA 연산기를 0.18um CMOS 공정으로 구현한 결과를 제시하고 기존의 연산기들과 그 성능을 비교하였으며, 1024-비트 지수승은 평균 365Kbps, 512-비트 지수승은 평균 1,233Kbps의 높은 성능을 가진다. 또한 제안하는 연산기는 부가적으로 모듈러 감산을 하드웨어적으로 수행할 수 있는데, 이는 몽고메리 매핑 계수를 계산하거나 중국인의 나머지 정리를 적용하기 위한 전처리 과정에 사용될 수 있다.

〔표 4〕. 기본 1024-비트 RSA 연산기들과의 성능 비교

	타겟 (um)	클락 사이클	주파수 (MHz)	게이트 카운트	연산 시간 (ms)	CRT/ nonCRT	모듈러 감산
제안하는 연산기	0.18	$1.5n^2/8$	300	192K	0.83	both	yes
Kwon 등 [13]	0.5	n^2	50	156K	43	nonCRT	no
Blum 등 [6]	FPGA	$n^2/2$	45	-	12	nonCRT	no
Cho 등 [22]	-	$n^2/2$	40	230K	13	nonCRT	no
McIvor 등 [23]	FPGA	$n^2/4$	97	-	2.73	CRT	no

참고문헌

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signature and public-key cryptosystems," Comm. ACM, vol. 21, pp. 120-126, 1978.

[2] G.R. Blakley, "A computer algorithm for the product ab modulo m," IEEE Trans. Comput., vol. C-32, pp. 497-500, 1983.

[3] E.F. Brickell, "A fast modular multiplication algorithm with application to two-key cryptography," Proc. CRYPTO'82 Advances Cryptology, pp. 51-60, 1982.

[4] P. L. Montgomery, "Modular multiplication without trial division," Math. Computation, vol. 44, pp. 519-521, 1985.

[5] Koc, C.K., Acar, T. Burton, and S. Kaliski Jr, "Analyzing and Comparing Montgomery Multiplication Algorithms," IEEE Micro., 16(3), pp. 26-33, June 1996.

[6] T Blum and C. Paar, "Montgomery modular exponentiation on reconfigurable hardware," Proc. 14th IEEE Symp. on Comp. Arith., pp. 70-77, 1999.

[7] N. Takagi, "A radix-4 modular multiplication hardware algorithm for modular exponentiation," IEEE Trans. Comput., vol. 41, pp. 949-956, Aug. 1992.

[8] P. Kornerup, "High-radix modular multiplication for cryptosystems," Proc. 11th IEEE Symp. Comput. Arith., Windsor, ON, Canada, pp. 277-283, June 1993.

[9] Jin-Hua Hong and Cheng-Wen Wu, "Cellular-Array Modular Multiplier for Fast RSA Public-Key Cryptosystem Based on Modified Booth's Algorithm" IEEE Trans. on VLSI Systems, vol. 11, no. 3, pp. 474-484, June 2003.

[10] C. Couvreur and J. J. Quisquater, "Fast decipherment algorithm for RSA public-key cryptosystem," Electronics letters, 18(21), pp. 905-907, 1982.

[11] Ciaran McIvor, Maire McLoone, and John V McCanny, "A high-speed, low latency RSA decryption silicon core," IEEE Inter. Symp. On Circuits and Systems(ISCAS), vol. 4, pp. 25-28, May 2003.

[12] Chung-Hsien Wu and Jin-Hua Hong, Cheng-Wen Wu, "RSA cryptosystem design based on Chinese remainder theorem," IEEE Proc. Asia and South Pacific Design Automation Conf.(ASP-DA), pp. 391-395, 2001.

[13] T. W. Kwon, C. S. You, W. S. Heo, Y. K. Kang, and J. R. Choi, "Two Implementation Methods of a 1024-bit RSA Cryptoprocessor Based on Modified Montgomery Algorithm," IEEE Inter. Symp. On Circuits and Systems(ISCAS), vol. 4, pp. 650-653, 2001.

[14] Ciaran McIvor, Maire McLoone, and John V McCanny, "Fast Montgomery Modular Multiplication and RSA Cryptographic Processor Architectures," 37th Asilomar Conference on Signals, Systems and Computers, vol. 1, no. 7, pp. 379-384, Nov. 2003.

[15] A. Cilardo, A.Mazzeo, L. Romano, and G.P. Saggese, "Carry-Save Montgomery Modular Exponentiation on Reconfigurable Hardware," IEEE Proc. on DATE'04, vol. 03, no. 3, pp. 206-211, 2004.

[16] T. Blum and C. Paar, "High-radix Montgomery modular exponentiation on reconfigurable hardware," IEEE Trans. Comput., vol. 50, issue 7, pp. 70-77, July 2001.

[17] Peter Kornerup, "Systolic, Linear-Array Multiplier for a Class of Right-Shift Algorithms," IEEE Trans. on Comp., vol. 43, issue 8, pp. 892-898, Aug. 1994.

[18] C. D. Walter, "Systolic modular multiplication," IEEE Trans. on Comp., vol. 42, issue 3, pp. 376-378, Mar. 1993.

[19] S. E. Eldridge and C. D. Walter, "Hardware

implementation of Montgomery's modular multiplication algorithm," IEEE Trans. on Comp., vol. 42, issue 6, pp. 693-699, June 1993.

[20] A. D. Booth, "A signed binary multiplication technique," Q. J. Mech. Appl. Math., vol. 4, issue 2, pp. 236-240, 1951.

[21] C. K. Koc and C. Y. Hung, "Fast algorithm for modular reduction," IEE Proc-Comput. Digit. Tech., vol. 145, no. 4, pp. 265-271, July 1998.

[22] Koon-Shik Cho, Je-Hyuk Ryu, and Jun-Dong Cho, "High-speed modular multiplication algorithm for RSA cryptosystem," IECON'01, pp. 479-483, 2001.

[23] Ciaran McIvor, Maire McLoone, and John V McCanny, "Modified Montgomery modular multiplication and RSA exponentiation techniques," IEE Proc-Comput. Digit. Tech., vol. 151, issue 6, pp. 402-408, Nov. 2004.

[24] 장남수, 임대성, 지성연, 김창한, 윤석봉, "고속 RSA 하드웨어 곱셈 연산과 하드웨어 구조," WISC 2006, pp. 31-41, Sep. 2006.

[25] 장남수, 임대성, 지성연, 윤석봉, 김창한, "고속 RSA 하드웨어 곱셈 연산과 하드웨어 구조," 정보보호학회논문지, 제17권 1호, pp. 11-20, 2007.

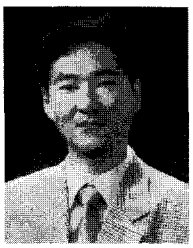
[26] 김무섭, 최용제, 김호원, 정교일, "개선된 몽고메리 알고리즘을 이용한 저면적용 RSA 암호 회로 설계," 정보보호학회논문지, 제12권 5호, pp. 95-105, 2002.

〈著者紹介〉

구 본 석 (Bonseok Koo) 정회원
1998년 2월 : 경북대학교 전자공학과 학사
2000년 2월 : 포항공과대학교 전자공학과 석사
2000년 2월~2000년 9월 : LG 정보통신 중앙 연구소 연구원
2000년 10월~현재 : 국가보안기술연구소 선임연구원
<관심분야> 암호칩 설계, 공개키 암호, 부채널 공격

유 권 호 (Gwonho Ryu) 정회원
1999년 2월 : 포항공과대학교 전자공학과 학사
2001년 2월 : 포항공과대학교 전자공학과 석사
2002년 9월~현재 : 국가보안기술연구소 연구원
<관심분야> 암호칩 설계, 블록 암호, 부채널 공격

장 태 주 (Taejoo Chang) 정회원
1982년 2월 : 울산대학교 전기공학과 학사
1990년 : 한국과학기술원 전기 및 전자공학과 석사
1998년 : 한국과학기술원 전기 및 전자공학과 공학박사
1982년~2000년 : 국방과학연구소 선임연구원
2000년~현재 : 국가보안기술연구소 책임연구원
<관심분야> 암호칩 설계, 정보보호, 통계학적 신호처리



이 상 진 (Sangjin Lee) 종신회원
1987년 2월 : 고려대학교 수학과 학사
1989년 2월 : 고려대학교 수학과 석사
1994년 2월 : 고려대학교 수학과 박사
1989년 2월~1999년 2월 : 한국전자통신연구원 선임연구원
1999년 2월~2001년 8월 : 고려대학교 자연과학대학 조교수
2000년 2월~현재 : 고려대학교 정보보호대학원 부교수
<관심분야> 대칭키 암호의 분석 및 설계, 정보은닉이론, 컴퓨터 포렌식