논문 2004-41SD-4-8

RNS(Residue Number Systems) 기반의 2,048 비트 RSA 설계

(Implementation of 2,048-bit RSA Based on RNS(Residue Number Systems))

권 택 원*, 최 준 림**

(Taek-Won Kwon and Jun-Rim Choi)

요 약

본 논문에서는 RNS(residue number systems) 몽고메리 모듈라 곱셈기 기반의 2,048 비트 RSA 설계를 제안한다. RNS는 긴 워드에 대한 모듈라 연산을 짧은 워드로 분할하여 고속 병렬 모듈라 연산을 처리하는 시스템으로써 본 논문에서는 RNS 몽고메리 모듈라 곱셈 연산을 위해 Wallace 트리 모듈라 곱셈기 기반의 Montgomery reduction method(MRM)[1]와 33개의 64 비트 RNS base 를 도입하였다. 또한, 고속 RNS 모듈라 곱셈 연산을 위해 Chinese remainder theorem(CRT)[2] 기반의 개선된 base extension 알고리즘을 제안한다. 본 논문에서 제시한 RNS 기반의 2,048 비트 RSA는 삼성 0.35㎛ 공정을 사용하여 기능을 검증하였으며 100㎒에서 2.53㎳ 연산 속도 결과를 얻었다.

Abstract

This paper proposes the design of a 2,048-bit RSA based on RNS(residue number systems) Montgomery modular multiplier. As the systems that RNS processes a fast parallel modular multiplication for a large word partitioned into small words, we introduce Montgomery reduction method(MRM)[1] based on Wallace tree modular multiplier and 33 RNS bases with 64-bit size for RNS Montgomery modular multiplication in this paper. Also, for fast RNS modular multiplication, a modified method based on Chinese remainder theorem(CRT)[2 is presented. We have verified 2,048-bit RSA based on RNS using Samsung 0.35㎛ technology and the 2,048-bit RSA is performed in 2.53㎳ at 100㎒.

Keywords: residue number systems(RNS), Montgomery reduction method(MRM), Chinese remainder theorem(CRT), Montgomery modular multiplication

I. 서 론

최근 전자서명을 위해 널리 사용되고 있는 RSA[3]암호 시스템은 안전도를 높이기 위해 키 값을 1,024 비트에서 2,048 비트로 확장하는 추세이며 이렇게 확장된 키 값으로 인해 반복적인 모듈라 곱셈 연산 구조를 갖는 하드웨어 구현을 매우 어렵게 만들고 있으며 좀 더 빠른 모듈라 곱셈기를 요구하고 있다. 지수승 연산의 결과는 모듈라 곱셈의 반복 연산을 통해 얻어지며[3-4] 모듈라 곱셈기를 어떻게 설계하느냐에 따라 전체 지수승의 처리 속도는 달라진다.

일반적인 모듈라 곱셈기로써 systolic 어레이 구조와 CSA 덧셈기 구조를 사용한 모듈라 곱셈기가 제안되었다.[5-6] 또한, 하드웨어 소모는 많지만 고속 모듈라 연산이 가능한 radix 기반의 몽고메리 모듈라 곱셈 알고리즘이 제안되었으며[7-8], 처리 속도는 느리지만 하드웨어 소모가 적은 분할형 워드 기반 모듈라 곱셈 알고리즘도 제안되고 있다.[9-10] 그러나, 이러한 기존의 방식들을 2048 비트 모듈라 곱셈 연산에 도입할 경우 2048 비트와 같은 긴 워드 길이에 대하여 모듈라 연산을 처리함

* 정회원, 삼성전자 반도체총괄
(Semiconductor Business, Samsung Electronics, Co., Ltd.)
** 정회원, 경북대학교 전자전기공학부
(School of Electrical Engineering and Computer Science, Kyungpook National University)
접수일자 : 2003년7월28일, 수정완료일: 2004년3월15일

으로써 파생되는 전파 지연의 문제를 갖고 있다. 이러한 전파 지연이 동작 주파수를 떨어지게 하여 전체 모듈라 곱셈 연산 속도를 저하시키는 원인이 된다. 본 논문에서는 이러한 긴 워드 길이로 발생되는 전파 지연 문제를 해결하고 2048 비트와 같은 긴 워드 길이에 대한 모듈라 곱셈을 고속으로 처리할 수 있는 RNS를 도입한다.[11-12]

RNS는 기본적으로 산술연산을 병렬로 처리하는 기법으로써 Montgomery 모듈라 곱셈 연산을 병렬로 처리할 수 있는 기법을 제공한다. 이러한 RNS(residue number systems)는 서로소의 관계를 갖는 moduli 원소들로 주어진 기저(base)에 대하여 어떤 정수를 각 moduli의 나머지(residue) 값으로 표현하는 방법을 말한다. 각각의 moduli 원소들은 독립적이며 연산 시 캐리 전파가 필요 없어 덧셈, 뺄셈, 혹은 곱셈 연산을 할 때 각각의 RNS 원소들의 계산을 독립적으로 처리할 수 있으며 여러 개의 프로세서를 두어 병렬처리를 할 수 있는 장점을 가진다.[2]

그러나, RNS를 RSA에 도입할 경우 몽고메리 모듈라 곱셈 연산시 base extension 연산을 수행하는 단점이 있다.[11-12] Base extension 연산은 기존의 RNS base에서 다른 RNS base로 확장하는 기법으로써 몽고메리 모듈라 곱셈 연산시 가장 복잡하고 긴 수행시간을 요구한다. 이러한 단점을 극복하기 위해 본 논문에서는 몽고메리 모듈라 곱셈 연산을 위해 Montgomery reduction method (MRM)를 도입하였으며 base extension 연산 시 요구되는 파라미터 k[11-12]를 고속으로 얻기 위해 Wallace tree[14] 구조를 사용했다.

먼저 본문 Ⅱ장에서는 RNS와 일반적인 RNS 기반의 몽고메리 모듈라 곱셈 알고리즘을 살펴보고 본문 Ⅲ장과 Ⅳ장에서는 RNS 기반의 몽고메리 모듈라 곱셈 알고리즘과 base extension 알고리즘을 제시하고 구조를 살펴본 후 기존의 RNS 기반의 RSA와 연산 속도 및 면적을 본문 Ⅴ장에서 비교, 검토하였다.

Ⅱ. RNS 기반의 몽고메리 모듈라 곱셈 알고리즘

1. Residue Number Systems(RNS)와 사칙연산

RNS는 정수(x)를 n 개의 서로 소(pair-wise relatively prime)의 관계를 가진 moduli(m)에 대하여 나머지 값(residue, $< x >_m$)으로 표현하는 방법으로 식 (1)과 같이 표현된다.

$$< x >_m = (< x >_{m_1}, < x >_{m_2}, \cdots, < x >_{m_n}) \quad (1)$$

여기에서 $< x >_{m_i} = x_i = x \bmod m_i$ 이며, 집합 $m = \{m_1, m_2, \cdots, m_n\}$은 RNS 기저(base)로써 크기가 n이며 $gcd(m_i, m_j) = 1 (i \neq j)$인 서로소의 관계이다. 이렇게 나머지 값($< x >_m$)으로 표현된 x는 CRT(Chinese remainder theorem)[2]에 의하여 식 (2)와 같이 원래의 정수 값으로 복원할 수 있다.

$$x = (\sum_{i=1}^{n} M_i \cdot < x_i \cdot \alpha_i >_{m_i}) \bmod M \quad (2)$$

여기에서 $M = \prod_{i=1}^{n} m_i$, $M_i = M/m_i$, 그리고 $\alpha_i = < M_i^{-1} >_{m_i}$이며 $x < M$의 관계에서 성립한다. RNS에서 사칙연산은 식 (3)과 (4)와 같이 덧셈, 뺄셈, 그리고 곱셈을 지원하며 캐리 전파가 없는 병렬 모듈라 연산 구조로 처리할 수 있다.

$$< x >_m \pm < y >_m = (x_1 \pm y_1, \cdots, x_n \pm y_n) \quad (3)$$

$$< x >_m \cdot < y >_m = (x_1 \cdot y_1, \cdots, x_n \cdot y_n) \quad (4)$$

2. RNS 몽고메리 모듈라 곱셈 알고리즘

RNS 몽고메리 모듈라 곱셈 알고리즘은 기존의 몽고메리 모듈라 곱셈 알고리즘을 RNS에서 계산할 수 있도록 변형한 알고리즘을 말하며 그림 1과 같다. 그림 1에서 RNS M($= \prod_{i=1}^{n} m_i$)과 RNS $\overline{M}$ ($= \prod_{i=1}^{n} \overline{m_i}$,)의 관계는 $GCD(M, \overline{M}) = 1$와 $M < \overline{M}$을 만족해야하며 RSA의 모듈러스 N과 M은 $0 < 2N < M$의 관계이다. 또한, 모듈라 곱셈기의 입력 A와 B는 모듈러스 N과 RNS M에 대하여 $A \cdot B < N \cdot M$의 관계를 항상 만족하여야 한다. 그림 1에서 step 1, 2a, 3, 4, 그리고 5a는 base m과 base $\tilde{m}$에 대하여 RNS 모듈라 곱셈과 덧셈 알고리즘을 사용하여 수행할 수 있다. Step 2b와 5b에서 base extension을 수행하는 이유는 일반적인 몽고메리 모듈라 곱셈 알고리즘의 step 4를 RNS 기반으로 수행할 경우 $v (= A \cdot B + t \cdot N)$가 항상 M의 배수가 되며 base m으로 표현할 경우 항상 0이 되어 base m으로 표현이 불가능해진다. 이러한 이유 때문에 base extension 연산을 수행하며 임의의 모듈러스 m_r을 사용하여 파라미터 k를 구하는 Shenoy-Kumaresan 방식[15]을 도

Montgomery Modular Multiplication		RNS Montgomery Modular Multiplication	
Input : A, B, N, R Output : $W=ABR^{-1}(modN)$		Input:$< A >_m$, $< A >_{\bar{m}}$, $< B >_m$, $< B >_{\bar{m}}$, $< N >_m$, $< N >_{\bar{m}}$ Output:$< W >_m$ ($W=ABM^{-1}(modN)$, $W<2N$)	
		Base m operation	Base $\bar{m}$ operation
step 1. $s \leftarrow A \cdot B$	→	step 1. $< s >_m \leftarrow < A >_m \cdot < B >_m$	$< s >_{\bar{m}} \leftarrow < A >_{\bar{m}} \cdot < B >_{\bar{m}}$
step 2. $t \leftarrow s \cdot (-N^{-1}) mod R$	→	step 2a.$< t >_m \leftarrow < s >_m \cdot < -N^{-1} >_m$	
		step 2b. $< t >_m \rightarrow < t >_{\bar{m}}$	
step 3. $u \leftarrow t \cdot N$	→	step 3.	$< u >_{\bar{m}} \leftarrow < t > \cdot < N >_{\bar{m}}$
step 4. $v \leftarrow s+u$	→	step 4.	$< v >_{\bar{m}} \leftarrow < s >_{\bar{m}} + < u >_{\bar{m}}$
step 5. $w \leftarrow v/R$	→	step 5a.	$< w >_{\bar{m}} \leftarrow < v >_{\bar{m}} \cdot < M^{-1} >_{\bar{m}}$
		step 5b. $< w >_m \leftarrow < w >_{\bar{m}}$	

그림 1. RNS 몽고메리 모듈라 곱셈 알고리즘[11-12]
Fig. 1. RNS Montgomery modular multiplication algorithm.[11-12]

입하면 base extension 연산을 수행할 수 있다.

식 (2)는 $0 \le x/M < 1$의 조건에서 정수 k가 유일하기 때문에 정수 k를 사용하여 다시 쓰면 식 (5)와 같이 되며 양변을 임의의 모듈러스 m_r에 대하여 나머지 연산을 하면 식 (6)과 같이 된다.

$$x = \sum_{i=1}^{n} M_i \cdot < x_i \cdot \alpha_i >_{m_i} - kM \tag{5}$$

$$< x >_{m_r} = < \sum_{i=1}^{n} M_i \cdot < x_i \cdot \alpha_i >_{m_i} - kM >_{m_r} \tag{6}$$

식 (6)을 k에 대하여 다시 쓰면 식 (8)이 되어 파라미터 k를 구할 수 있다.

$$y = \sum_{i=1}^{n} M_i \cdot < x_i \cdot \alpha_i >_{m_i} - < x >_{m_r} \tag{7}$$

$$k = << M^{-1} >_{m_r} \cdot y >_{m_r} \tag{8}$$

파라미터 k를 구한 후 식 (9)를 사용하여 base m에서 base $\bar{m}$으로 base extension 연산을 수행할 수 있다.

$$< x >_{\bar{m}} = < \sum_{i=1}^{n} M_i \cdot < x_i \cdot \alpha_i >_{m_i} - kM >_{\bar{m}} \tag{9}$$

그러나, 이러한 Shenoy-Kumaresan 방식은 파라미터 k를 구하기 위해 식 (7)에서 $< x >_{m_r}$을 구해야하는데 $< x >_{m_r}$은 식 (2)를 사용하여 정수 x를 구한 후 임의의 모듈러스 m_r에 대하여 나머지 연산을 해야 함으로 실제 연산 시 많은 시간이 소모되는 단점이 있다. 본 논문에서는 이러한 단점을 해결하기 위해 최적의 base m을 선택하여 k와 base extension을 고속으로 처리할 수 있는 기법을 제안한다.

Ⅲ. MRM 기반의 RNS 몽고메리 모듈라 곱셈 알고리즘과 Base Extension

1. MRM 기반의 RNS 몽고메리 모듈라 곱셈 알고리즘[11]

Montgomery reduction method(MRM)는 몽고메리 모듈라 곱셈 알고리즘을 하드웨어로 쉽게 구현하기위한 기법이다. 그림 2와 같이 MRM 기법은 r번 수행하여 왼쪽으로 $R(=2^r)$만큼 쉬프트하면서 나머지 연산을 수행하는 알고리즘이다. 따라서 하드웨어로 구현할 경우 CSA 어레이로 쉽게 구현이 가능한 장점을 지닌다.[6] 본 논문에서는 MRM을 RNS 몽고메리 곱셈 알고리즘에 적용하였으며 그림 1의 RNS 몽고메리 모듈라 곱셈 알고리즘은 그림 3과 같이 변형된다. 그림 1의 step 1에서 5b까지의 모든 모듈라 연산의 결과는 MRM을 적용했을 때 그림 3에서와 같이 $< R^{-1} >_m$항이 곱해진 형태로 결과 값이 나타난다. 이와 같이 MRM을 적용할 경우 base m과 base $\bar{m}$에 대하여 모듈라 연산을 그림 2와 같이 쉬프트 연산으로 간단히 처리할 수 있는 장점뿐만 아니라 하드웨어 구현을 CSA 어레이로 쉽게 구성할 수 있다. 그러나, RNS 몽고메리 모듈라 곱셈 연산에서는

Input : A, B(residue), m_i(modulus) Output : $y = < A \cdot B \cdot R^{-1} >_{m_i}$ $(R = 2^r)$
step 1. $y = 0$ step 2. for $j = 0$ to $r-1$ do step 2a. $y = y + A_i \cdot B$ step 2b. $y = y + y_0 \cdot m_i$ // modular 연산 step 2c. $y = y/2$ // left shift step 3. end for

그림 2. Bit-level MRM (x_j : x의 j번째 bit)[11]
Fig. 2. Bit-level MRM (x_j : j-th bit of x)[11]

Function : $< x >_{\bar{m}} = BE1 (< x >_m)$
Input : $< x >_m$, Output : $< x >_{\bar{m}}$ Precomputation(constant): $< R\alpha >_m$, $< R^4 M_i >_{\bar{m}}$
step 1. $\xi_i = < x >_{m_i} \cdot < R\alpha_i >_{m_i} \bmod m_i$ step 2. $y_{i0} = 0$ step 3. For $j = 1, \cdots, n$ step 4. $y_{ij} = y_{i(j-1)} + \xi_{(i+1-j)} \cdot < R^4 M_{(i+1-j)} >_{\bar{m}}$ step 5. Next j step 6. $< x >_{\bar{m}} = y_{in} \bmod \bar{m}$

그림 4. 그림 3의 Base extension 1 알고리즘
Fig. 4. Base extension 1 algorithm in Fig. 3.

RNS Montgomery Modular Multiplication Based on MRM	
Input:$< A >_m$, $< A >_{\bar{m}}$, $< B >_m$, $< B >_{\bar{m}}$, $< N >_m$, $< N >_{\bar{m}}$	
Output:$< W >_m$ $(W = ABM^{-1} (mod N)$, $W < 2N)$, $R = 2^r$	
Base m operation	Base $\bar{m}$ operation
step 1. $< s >_m \leftarrow < A >_m \cdot < B >_m \cdot < R^{-1} >_{m_i}$	$< s >_{\bar{m}} \leftarrow < A >_{\bar{m}} \cdot < B >_{\bar{m}} \cdot < R^{-1} >_{\bar{m}}$
step 2a. $< t >_m \leftarrow < s >_m \cdot < N^{-1} >_m \cdot < R^{-1} >_{m_i}$	
step 2b. $< t >_m \rightarrow < t >_{\bar{m}}$ (Base Extension 1 : BE1)	
step 3.	$< u >_{\bar{m}} \leftarrow < t >_{\bar{m}} \cdot < N >_{\bar{m}} \cdot < R^{-1} >_{\bar{m}}$
step 4.	$< v >_{\bar{m}} \leftarrow (< s >_{\bar{m}} + < u >_{\bar{m}}) \cdot < R^{-1} >_{\bar{m}}$
step 5a.	$< w >_{\bar{m}} \leftarrow < v >_{\bar{m}} \cdot < RM^{-1} >_{\bar{m}} \cdot < R^{-1} >_{\bar{m}}$
step 5b. $< w >_m \leftarrow < w >_{\bar{m}}$ (Base Extension 2 : BE2)	

그림 3. MRM 기반의 RNS 몽고메리 모듈라 곱셈 알고리즘[11]
Fig. 3. RNS Montgomery modular multiplication algorithm based on MRM[11]

모듈라 덧셈 연산과 base extension 연산을 포함하므로 step 5b의 연산 결과에서 reduction 변수 R항을 없앤 $< w >_m$ 값을 얻도록 step 2b와 5b에서 base extension 알고리즘을 변형해야한다. 즉, step 2a, 4, 5a의 연산 결과는 $< (R^2)^{-1} >_m$항을 포함한 결과를 얻게 되며 step 2b와 5b의 base extension 연산에서 reduction 변수 R이 포함된 상수 값 α_i와 M_i값을 곱하여 $< (R^2)^{-1} >_m$ 항을 제거한 $< w >_m$값을 얻도록 한다.

2. 개선된 Base Extension 알고리즘

Base extension 연산은 Shenoy-Kumaresan의 옵셋 결과[13]에 의하여 그림 3의 step 2b에서는 파라미터 k를 구할 필요가 없으며 step 5b에서는 base extension에서 발생되는 옵셋을 제거하기 위해 파라미터 k를 구해야한다. Step 2b의 base extension 연산은 CRT 알고리즘을 사용하여 base m에 대하여 그림 4의 step 1과 같이 $< t_i \cdot R\alpha_i >_{m_i}$을 모두 병렬로 구한 후 step 4에서 그 값에 $< R^4 M_i >_{\bar{m}_i}$을 곱하여 base $\bar{m}$에 대하여 나머지 연산을 수행한다. 그 결과를 모두 더한 후 step 6에서 base $\tilde{m}$에 대하여 나머지 연산을 수행하면 그림 4에서와 같이 그림 3의 step 2b에서의 t를 base m에서 base $\bar{m}$로 base extension할 수 있다.

그림 4에서 모든 나머지 연산은 MRM을 사용한 것으로써 step 1에서 상수 값으로 R을 포함한 $< R\alpha_i >_{m_i}$과 그림 4의 step 4에서 R^4을 포함한 $< R^4 M_i >_{\bar{m}}$ 상수를

Function : $< x >_m = BE2(< x >_{\overline{m}})$
Input : $< x >_{\overline{m}}$,
Output : $< x >_m$
Precomputation(constant) : $< R\alpha >_{\overline{m}}$, $< R\alpha >_m$
$< R^4 \overline{M_i} >_m$, $< R(-\overline{M}) >_m$
step 1. $< x >_m = BE1(< x >_{\overline{m}})$
step 2. $k = 0$
step 3. For $i = 1, \cdots, n$
step 4. $k = k + < \overline{x_i} \cdot R\alpha_i >_{\overline{m_i}}$
step 5. Next i
step 6. $k = (k + 2^{r-1})/2^r$
step 7. $< x >_{m_i} = x_i + < k \cdot < R(-\overline{M}) >_{m_i} >_{m_i}$
step 8. $< x >_{m_i} = < x >_{m_i} \cdot < R^2 >_{m_i}$

그림 5. 그림 3의 Base extension 2 알고리즘
Fig. 5. Base extension 2 algorithm in Fig. 3.

곱한 이유는 step 6에서 reduction 변수 R을 포함하지 않는 $< x >_{\overline{m}}$값을 얻기 위해서다.

그림 3에서 base extension 2는 옵셋 $k\overline{M}$[15]값을 제거하기 위해 파라미터 k를 구해야한다. 파라미터 k는 식 (5)에서 양변에 M을 나누어 주면 $0 \le x/M < 1$와 $0 < \xi_i/m_i < 1$의 성질을 이용하여 식 (10)을 얻을 수 있다.

$$k = \sum_{i=1}^{n} \frac{\xi_i}{m_i} \tag{10}$$

식 (10)에서 base m을 $m_i = 2^r - \mu_i (2^r \gg \mu_i)$로 설정할 경우 식 (10)에서 ξ_i/m_i는 식 (11)과 같이 확장할 수 있다.

$$\begin{aligned} \sum_{i=1}^{n} \frac{\xi_i}{m_i} &= \sum_{i=1}^{n} \frac{\xi_i}{2^r - \mu_i} \\ &= \sum_{i=1}^{n} \frac{\xi_i}{2^r} \cdot \frac{1}{1 - \mu_i 2^{-r}} \\ &= \frac{1}{2^r} \cdot \sum_{i=1}^{n} (\xi_i + \xi_i \mu_i 2^{-r} + \xi_i \mu_i^2 2^{-2r} + \cdots) \end{aligned} \tag{11}$$

이 때, 식 (10)에서 m_i가 2^r로 근사할 경우 식 (10)의 base extension factor k는 식 (12)로 근사할 수 있다.

$$k = \frac{\sum_{i=1}^{n} \xi_i}{2^r} \tag{12}$$

식 (12)와 같이 근사할 경우 오차는 식 (13)과 같으며 오차 최대값은 $\xi_i < 2^r$, $\mu_i < 2^l$, $r > l$ 그리고 $n \le 2^q$의 조건에 의해서 $2^{-(r-l-q-1)}$이다.

$$\begin{aligned} e_k &= \sum_{i=1}^{n} \xi_i/m_i - \sum_{i=1}^{n} \xi_i/2^r \\ &= \frac{1}{2^r} \sum_{i=1}^{n} \xi_i \mu_i 2^{-r} (1 + \mu_i 2^{-r} + \mu_i^2 2^{-2r} + \cdots) \\ &< \sum_{i=1}^{n} 2^{-(r-l-1)} \\ &= n \cdot 2^{-(r-l-1)} \\ &\le 2^{-(r-l-q-1)} = E_k \end{aligned} \tag{13}$$

그러므로, 식 (12)는 식 (14)에 의하여 k 혹은 k-1의 값을 가질 수 있다.

$$k + x/M - E_k < \sum_{i=1}^{n} \xi_i/2^r \le k + x/M \tag{14}$$

본 논문에서는 식 (12)가 항상 정확한 k값을 가질 수 있도록 $E_k \le \beta < 1$을 만족하는 $\beta(=1/2)$를 식 (12)에 삽입하여 분모 근사화로 발생하는 오차를 보상하였으며 식 (12)는 식 (15)로 변형된다.

$$k = \frac{\sum_{i=1}^{n} \xi_i + 2^{r-1}}{2^r} \tag{15}$$

따라서 식 (15)와 그림 4의 BE1을 이용하면 그림 5와 같이 base $\overline{m}$에서 base m으로 base extension 2(BE2)를 수행할 수 있다. 그림 5에서 BE2는 먼저 그림 4의 BE1 알고리즘을 이용하여 base m으로 표현한 후 식 (11)의 파라미터 k를 구하기 위해 step 3에서 6까지 수행한다. Step 7에서 step 6에서 구한 k를 입력하여 식 (9)의 연산을 수행한다. 이 때 step 7의 연산 결과는 $< R^{-1} >_{m_i}$항을 포함하므로 이 항을 없애기 위해 step 8을 수행하여 그림 3의 base m에 대한 $< w >_m$ 값을 얻는다. 본 논문에서 제시한 base extension 연산은 2,048 비트 RSA 연산을 위해 33개의 64 비트 RNS base를 사용했다고 가정했을 때 표 1과 같이 기존의 [11-12] 방식보다 연산에 필요한 연산시간이 줄어들어 전체 RNS 모듈라 곱셈 연산 속도를 향상시키는 장점이 있다. 표 1의 [11]에서 제시한 방식은 본 논문에서 제안한 MRM을 사용하였지만 모듈라 덧셈 연산시 ALU의

표 1. Base extension 연산시 필요한 클럭 수 비교
Table 1. Comparison of clocks for base extension

BE	Ref. [11]	Ref. [12]	Proposed
BE1	68	66	35
BE2	122	66	38

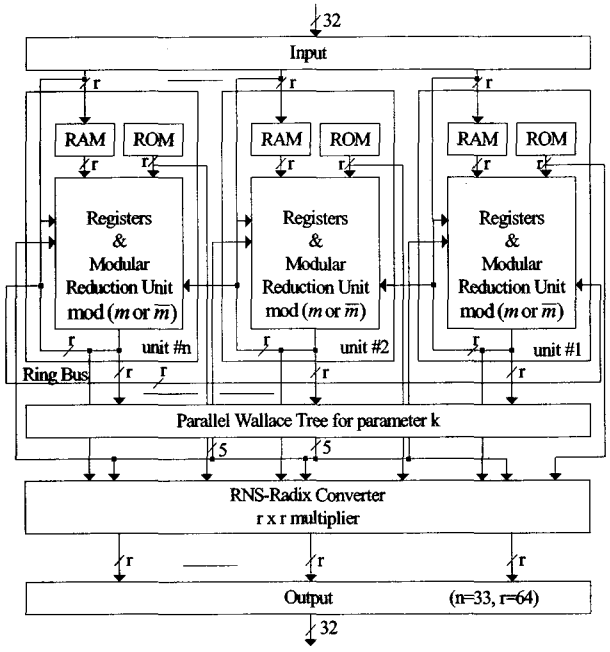


그림 6. RNS 몽고메리 모듈라 곱셈 연산을 위한 병렬 구조
Fig. 6. Parallel architecture for RNS Montg mery modular multiplication.

나눗셈기를 사용하여 연산 속도를 크게 떨어뜨리는 원인이 되었다. 또한, [12]에서는 MRM 방식을 사용하지 않고 내부에 곱셈기를 두어 모듈라 연산을 처리하므로써 기본적으로 연산시간이 다소 증가한 원인이 되었다.

Ⅳ. RNS 몽고메리 모듈라 곱셈기 기반의 2,048 비트 RSA 구현

본 논문에서는 2,048 비트 모듈라 곱셈을 RNS 기반으로 처리하기 위해 33개의 64(r) 비트 RNS base를 두어 그림 6과 같이 33(n)개의 unit으로 병렬연산을 한다. 각 병렬 unit 내부에는 base extension을 위해 상수 값을 저장하기 위한 ROM과 RNS 몽고메리 모듈라 곱셈 연산 결과를 저장하기 위한 RAM으로 구성된다. 또한, base $m(2^r-\mu,\ \mu \ll 2^r)$과 base $\overline{m}$에 대하여 나머지 연산을 위한 MRM 기반의 modular reduction unit이 있으며 나머지 연산의 결과를 저장하기 위한 임시 레지스터

표 2. Base extension 연산을 위한 ROM1 테이블
Table 2. ROM1 table for base extension.

	Parameters
BE1	$< M_i^{-1} >_{m_i},\ < M_1 >_{\overline{m}}, \cdots, < M_n >_{\overline{m}}$
BE2	$< \overline{M}_i^{-1} >_{\overline{m}},\ < \overline{M}_1^{-1} >_{m_i}, \cdots, < \overline{M}_n^{-1} >_{m_i},$ $<-\overline{M}>_{m_i}$

표 3. Radix-RNS 변환과 RNS-Radix 변환을 위한 ROM2 테이블
Table 3. ROM2 table for Radix-RNS and RNS-Radix conversion.

	Parameters
$x \rightarrow < x >_m$	$< 2^r >_{m_i},\ < 2^{r2} >_{m_i}, \cdots, < 2^{r(n-1)} >_{m_i}$
$x \rightarrow < x >_{\overline{m}}$	$< 2^r >_{\overline{m}},\ < 2^{r2} >_{\overline{m}}, \cdots, < 2^{r(n-1)} >_{\overline{m}}$
$< x >_m \rightarrow x$	$M_1, \cdots, M_n,\ -M$

로 구성된다. 33개의 병렬 unit들은 base extension을 고속으로 처리하기 위하여 ring bus[12]로 연결되어있으며 base extension 연산 시 ring bus를 통해 그림 4와 5의 ξ_i값을 n번 회전하여 각 unit에 전달된다. 그림 5의 파라미터 k를 고속으로 구하기 위하여 입력이 각각 17개와 16개를 갖는 Wallace tree를 병렬로 두어 처리하며 연산결과를 병렬 unit으로 전달된 후 그림 5의 base exten -sion 연산을 수행한다. 지수만큼 반복 모듈라 연산을 모두 수행한 후 그림 6의 RNS-Radix 변환기를 거쳐 최종 모듈라 지수승 값을 얻게된다.

RNS 몽고메리 모듈라 곱셈기는 그림 3의 step 1에서 5b까지 모듈라 곱셈, 덧셈, 그리고 base extension 연산을 수행하기 위하여 내부에 base m과 base $\overline{m}$에 대한 MRM 기반의 modular reduction unit이 있으며, 표 2와 같이 base extension 연산과 표 3과 같이 RNS-Radix 혹은 Radix-RNS 변환[12]을 위한 파라미터 저장을 위해 ROM 테이블이 있다. 그리고, 그림 4의 step 4에서 덧셈연산을 하기 위하여 72 비트 덧셈기와 레지스터(R1)를 포함하고 있다. 그림 4의 base extension 1(BE1) 연산 시 step 1의 ξ_i는 그림 7의 각 unit에서 계산되어 레지스터 R2에 저장된 후 $j=1$일 때 step 4의 연산을 수행 후 그 결과를 레지스터 R1에 저장을 한다. 그 다음 연산은 $j=2$를 연산하기 전에 레지스터 R2에 그 이전의 unit에 의해 계산된 ξ_{i-1}를 저장하고 $j=2$를 연산한다. 이렇게 n번 수행하면 그림 4의 y_{in}값이 레지스터 R1에 저장되고 base $\overline{m}$에 대하여 나머지 연산을 수행하면

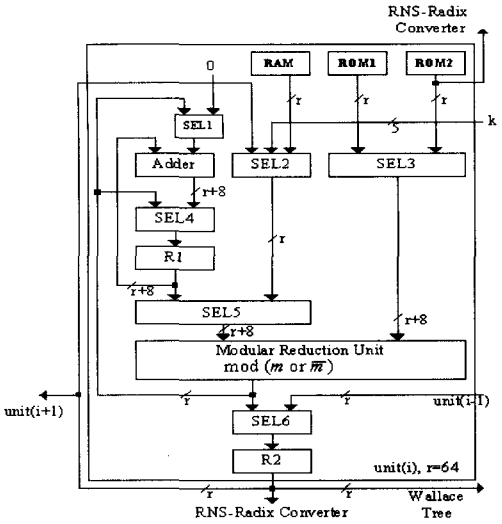
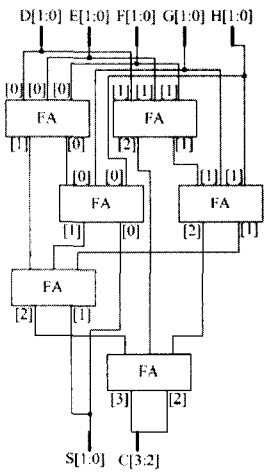
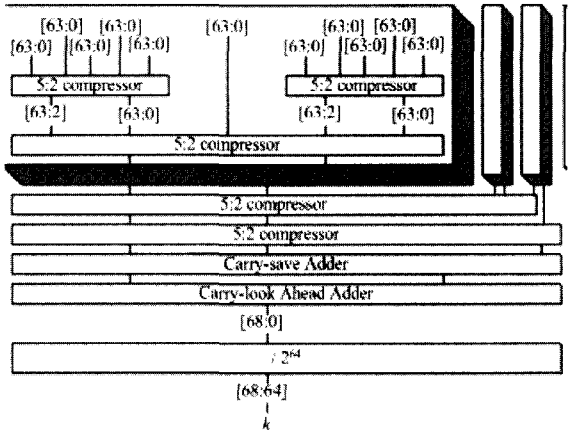


그림 7. RNS 몽고메리 모듈라 곱셈기
Fig. 7. RNS Montgomery modular multiplier.



(a) 2-bit 5:2 compressor



(b) 34-input Wallace tree adder

그림 8. 파라미터 k를 위한 Wallace tree adder
Fig. 8. Wallace tree adder for parameter k

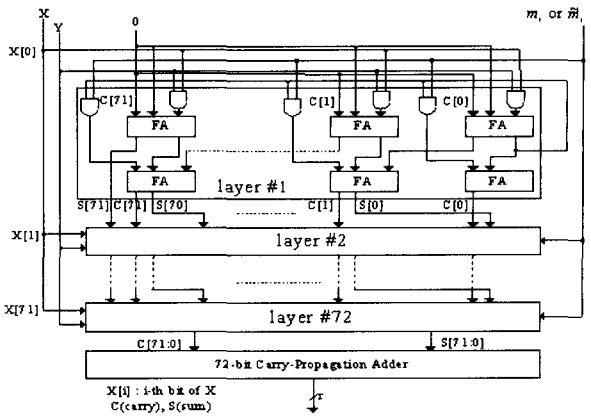


그림 9. MRM 기반의 modular reduction unit
Fig. 9. Modular reduction unit based on MRM

표 4. 각 블록의 게이트 수
Table 4. Gate counts for each block.

Blocks		Gate Counts
RNS Modular Multiplier	register	42K
	ROM (87K)	28K
	RAM (2K)	8K
	MUX	5K
	Modular reduction unit	105K
Multiplier		9K
Wallace Tree		5K
Buffer		12K
Controller		10K
I/O		3K
Total		227K

base m에 대한 나머지 값을 base $\overline{m}$에 대한 나머지 값으로 base extension 1을 수행할 수 있다. 그림 5의 BE2는 BE1을 수행한 후 step 3에서 6을 그림 8의 병렬 Wallace tree로 한 번에 파라미터 k를 계산하여 step 7과 8에서 BE2를 수행한다. 그림 8의 병렬 Wallace tree는 64 비트 길이의 입력 33개를 한 번에 더하여 69비트 길이의 출력을 갖는 고속 덧셈기로써 지연시간은 8번의 CSA와 2번의 CPA 연산시간이다.

Modular reduction method(MRM) 기반의 modular reduction unit은 그림 9와 같다. 그림 9의 모듈라 연산기는 부가적으로 필요한 비교기를 없애기 위해[1] 72비트 입력 X, Y에 대하여 64 비트 $X \cdot Y \cdot 2^{-72} \bmod (m \text{ or } \overline{m})$ 출력을 갖는다. 72 비트 입력에 대하여 나머지 연산을 한 번에 수행하기위해 CSA 어레이는 72단으로 구성하였으며 72-bit CPA 덧셈기로 캐리와 합을 최종 더하여 64 비트 길이의 나머지 값을 출력한다. 72번의 쉬프트 연산을 수행하는 72단의 CSA 어레이는 그림 1과 같은 bit-level 몽고메리 모듈라 곱셈 연산을 한 번에 처리함으로써 연산속도를 높일 수 있다.

표 5. RSA의 연산 속도 비교
Table 5. Comparison of the speed of RSA.

구조		RSA [bits]	주파수 [㎒]	연산 속도 [㎳]
systolic	Ref. [16]	2,048	50	167.7
	Ref. [17]	2,048	50	83.9
CSA	Ref. [6]	2,048	50	85.4
RNS	Ref. [11]	1,024	50	8.2
	Ref. [12]	2,048	80	29.2
	Proposed	2,048	100	2.53
상용칩	Ref. [18]	1,024	50	5.25
	Ref. [19]	1,024	?	5

V. 제안한 RNS 몽고메리 모듈라 곱셈기의 구현 및 연산 속도 비교

지금까지 제시한 RNS 몽고메리 모듈라 곱셈기를 기반으로 2,048 비트 RSA 연산기를 삼성 0.35㎛ 공정을 사용하여 구현하였다. 33개의 64 비트 RNS base$m$ or $\overline{m}$에 대하여 33개의 병렬 구조로 설계하였으며 표 4와 같이 약 227K의 게이트를 사용하였다. 본 논문에서 제안한 RNS 기반의 RSA는 RNS-Radix 변환에만 부가적으로 곱셈기를 사용하였으며 전체 면적에서 4%의 증가만을 가져왔다.

본 논문에서 제안한 2,048비트 RNS 기반 RSA의 연산 속도를 여러 구조를 사용한 기존의 RSA와 상용 RSA 칩과 표 5에서 비교하였다. 본 논문에서 제안한 RSA는 0.35㎛ 공정으로 설계시 연산 속도가 향상됨을 알 수 있었다.

VI. 결 론

본 논문에서는 네트워크 보안을 위한 RSA 시스템의 핵심 연산블록인 2,048 비트의 RSA 모듈라 지수승 프로세서를 RNS 몽고메리 모듈라 곱셈기 기반으로 설계하였으며 성능을 비교 검토하였다. 본 논문에서 제안한 RNS 몽고메리 모듈라 곱셈기는 Montgomery reduction method(MRM) 기반의 모듈라 곱셈기로 구성되었으며 Wallace tree 덧셈기와 링 버스 구조를 사용하여 RNS base extension 연산을 고속 처리 할 수 있었다. 모듈라 곱셈 연산을 한 번 처리하는데 78 클럭을 소모하였으며 2,048 비트 지수를 처리하는데 L-R 이진 지수승 방식을 사용하여 평균적으로 $1.5 \times 2,048 \times 78$ 클럭을 소모하였으며 100㎒에서 2.53㎳ 연산 속도를 보였다.

참 고 문 헌

[1] P. Montgomery, "Modular multiplication without trial division," Mathematics of Computation, vol. 44, no. 170, pp. 519-521, April 1985.

[2] P. V. Ananda, Residue number systems: algorithms and architectures, Kluwer academic publishers, 2002.

[3] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," Communications of the ACM, vol. 21, no. 2, pp. 120-126, Feb. 1978.

[4] C. K. Koc, "RSA Hardware Implementation," RSA Lab., Technical Report TR-801, v1.0, Aug. 1995.

[5] C. K. Koc and C. Y. Hung, "Bit-level systolic arrays for modular multiplication," Journal of VLSI Signal Processing, vol. 3, no. 3, pp. 215-223, Oct. 1991.

[6] T. W. Kwon, J. R. Choi and etc., "Two imple mentation methods of a 1024-bit RSA cryptopro cessor based on modified Montgomery algorithm," Circuits and Systems, ISCAS 2001, vol. 4, pp. 650 -653, Sydney, 2001.

[7] T. Blum and C. Paar, "High-radix Montgomery modular exponentiation on reconfigurable hardwa re," IEEE Trans. on Computers, vol. 50, no. 7, pp. 759-764, May 2001.

[8] 권택원, 최준림, "Radix-2k 모듈라 곱셈 알고리즘 기반의 RSA 지수승 연산기 설계," 한국정보보호학회논문집, 제12권, 제2호, 35-43쪽, 2002년 4월.

[9] F. Tenca and C. K. Koc, "A scalable architecture for Montgomery multiplication," CHES1999, LNCS 1717, pp. 94-108, Springer-Verlag, Aug. 1999.

[10] 권택원, 최준림, "가상 캐리 예측 덧셈기와 PCI 인터페이스를 갖는 분할형 워드 기반 RSA 암호 칩의 설계," 대한전자공학회논문집, 제39권, SD편, 제8호, 34-41쪽, 2002년 8월.

[11] B. J. Phillips, "Montgomery residue number syste ms," Electronics Letters, vol. 37, no. 21, pp. 1286 -1287, Oct. 2001.

[12] N. Hanae, M. Masahiko, S. Atsushi, and K. Shini

chi, "Implementation of RSA algorithm based on RNS Montgomery multiplication," CHES 2001, LNCS 2162, pp. 364-376, Springer-Verlag, May 2001.
[13] N. S. Szabo and R. I. Tanaka, Residue arithmetic and its application to computer technology, New York, McGraw-Hill, 1967.
[14] P. Behrooze, Computer arithmetic: algorithms and hardware designs, Oxford, 2000.
[15] A. P. Shenoy and R. Kumaresan, "Fast base extension using a redundant modulus in RNS," IEEE Trans. on Computers, vol. 38, no. 2, pp. 292-297, Feb. 1989.
[16] C. Y. Su, S .A. Hwang, P. S. Chen, and C. W. Wu, "An improved Montgomery algorithm for high-speed RSA public-key cryptosystem," IEEE Trans. on VLSI Systems, vol. 7, no. 2, pp. 280-284, June 1999.
[17] T. Blum and C. Paar, "High-radix Mon- tgomery exponentiation on reconfigurable hardware," IEEE Trans. on Computers, vol. 50, no. 7, pp. 759-764, April 2001.
[18] Hi/fn Company, Hi/fn 6500, http://www.hifn.com/docs/6500.pdf.
[19] http://www.rainbow.com/crytoswift.

저 자 소 개

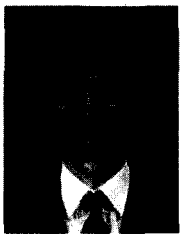
권 택 원(정회원)

1998년 경북대학교 전자공학과 학사

2000년 경북대학교 전자공학과 석사

2004년 경북대학교 전자공학과 박사

2004년 2월 ~ 현재 삼성전자 반도체총괄 연구원



최 준 림(정회원)

1986년 연세대학교 전기공학과 학사

1988년 미국 Cornell 대학교 전자전기공학과 석사

1991년 미국 Minnesota 대학교 전자전기 공학과 박사

1991년 7월 ~ 1997년 2월 LG 전자기술원 책임 연구원

1997년 3월 ~ 현재 경북대학교 전자전기컴퓨터학부 부교수