Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.27, NO.2, Apr. 2017 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2017.27.2.193

Modular 연산에 대한 오류 탐지*

김 창 한,[1†] 장 남 수[2‡]
[1]세명대학교, [2]세종사이버대학교

Error Detection Architecture for Modular Operations*

Chang Han Kim,[1†] Nam Su Chang[2‡]
[1]Semyung University, [2]Sejong Cyber University

요 약

본 논문에서는 정수 모듈러 N(홀수) 연산을 모듈러 $(2^r-1)N$ 연산으로 변환하여 연산중 발생하는 오류를 탐지하는 방법을 제시한다. 제안하는 방법은 모듈러 직렬 곱셈기의 경우 공간 복잡도는 50% 정도, 시간 복잡도는 1% 미만 증가한다, 제안하는 방법은 r=2 인 경우 1 비트 오류는 99%, 2 비트 오류는 50%, r=3 인 경우 1, 2 비트 오류를 99% 탐지가 가능한 효율적인 오류 탐지 방법이다.

ABSTRACT

In this paper, we proposed an architecture of error detection in Z_N operations using $Z_{(2^r-1)N}$. The error detection can be simply constructed in hardware. The hardware overheads are only 50% and 1% with respectively space and time complexity. The architecture is very efficient because it is detection 99% for 1 bit fault. For 2 bit fault, it is detection 99% and 50% with respective r=2 and r=3.

Keywords: Modular Operation, Error Detection, Bit-Serial Multiplier

I. 서 론

정보보호분야에서 사용하는 많은 프로토콜은 정수 모듈러 연산에 바탕을 두고 있다[3,4,5]. RSA 관련 암호는 합성수 N, 타원곡선, Diffie-Hellman, DSA 등은 소수 P에 바탕을 둔 모듈러 연산을 사용한다[4,5].

이 같은 관계로 1980년 이후 모듈러 연산의 효율적인 소프트웨어, 하드웨어 구현을 위한 연구[15, 16,17]가 활발하게 진행되었다. 2000년대부터 모든 암호에 오류주입 공격이 등장하면서 이를 방어하기 위한 다양한 방안이 연구되고 있다[6]. 연산중 발생하는 오류를 탐지하는 방법도 이러한 연구 분야중 하나이다. 특히 CRT RSA의 경우 폴트 공격[14]에 매우 취약하다. 또한 큰 정수 연산의 경우 다양한 이유로 연산 중 오류가 발생할 수 있으며 이러한 오류를 탐지하는 방법을 1988년에 T.J. Brosnan and N.R. Strader II[9]는 bit-Serial 정수 곱셈에 대하여 *Modular* $s(s=2^r-1)$를 이용한 아주 간결한 방법을 제안하였다. 그러나 그 후 뚜렷한 연구가 없었으나 2014년에 암호에 활용하는 모듈러 곱셈 방법 중 가장 많아 사용하는 몽고메리 곱셈에 대한 오류 탐지 방법이 G. Zervakis 등[10]에 의하여 제안되었다. 본 논문에서는 1988년에 제안된 정수 곱셈의 오류 탐지 방법을 활용하여 암호분야에서 널리 활용되는 일반적인 모듈러 N 연산의 오류를 탐지할 수 있는 방법을 제시한다. 2장에서는 오류 탐지 방법의 기본적인 이론을 설명하였으며 3장에서는 이를 모듈러 직렬곱셈기에서 처리하는 구체적인 방법을, 4

Received(02. 23. 2017), Modified(03. 23. 2017), Accepted(03. 23. 2017)
* 이 논문은 2015학년도 세명대학교 교내학술연구비 지원에 의해 수행된 연구임
† 주저자, chkim@semyung.ac.kr
‡ 교신저자, nschang@sjcu.ac.kr(Corresponding author)

장에서는 공간 및 시간 복잡도를 분석하였다. 제안한 모듈러 오류 탐지 방법은 $s=3$인 경우 전체적으로 공간복잡도 50%, 시간복잡도 1% 이하의 추가 부담이 필요하고 1비트 오류는 99%, 2비트 오류는 50%가 탐지 가능하다. 특히 $s=7$인 경우는 공간 복잡도는 54%정도 시간 복잡도는 1%정도 증가하나 1, 2비트 오류를 99% 탐지 가능한 강력한 탐지 능력을 갖는다.

II. 기존 정수 곱셈의 오류 탐지

1988년에 T.J. Brosnan and N.R. Strader II [9]는 bit-Serial 정수 곱셈에 관한 오류 탐지 방법을 제안하였다. 이 방법은 *Modular* 2^r-1을 이용하는 방법이다. 즉, $r=2$인 경우, 두 정수를

$$A=(\mathrm{A_{n-1}},\mathrm{A_{n-2}},\cdots,\mathrm{A_1},\mathrm{A_0})_{2^2},$$
$$B=(\mathrm{B_{m-1}},\mathrm{B_{m-2}},\cdots,\mathrm{B_1},\mathrm{B_0})_{2^2}$$

라 할 때

$$C=\mathrm{AB}=(\mathrm{C_{n+m-2}},\mathrm{C_{n+m-3}},\cdots,\mathrm{C_1},\mathrm{C_0})_{2^2}$$

라 하면

$$C\mathrm{Mod3}=\sum_{i=0}^{k-1}\mathrm{C_i}\,\mathrm{Mod3}$$
$$=\mathrm{AB}\,\mathrm{Mod3}$$
$$=(\sum_{i=0}^{n-1}\mathrm{A_i})(\sum_{j=0}^{m-1}\mathrm{B_j})\mathrm{Mod3} \qquad (1)$$

이다.

식 (1)에서 A, B를 표현하는 모든 비트들이 *Modular* 3으로 표현되며 그 곱셈 또한 표현되는 것을 이용하여 모든 정수 곱셈의 오류를 1/3의 확률로 확인할 수 있는 방법이다. 기본 구조는 Fig. 1과 같다.

Fig.1의 *Mod 3 generator*에서 *Mod 3 Adder*가 필요하며, 이 *Adder*는 일반적인 *Mod 3 Adder*와는 달리

$$(0,0),\ (0,1),\ (1,0)\ (1,1)$$

를 입력 값으로 받는 *Adder*이다. 이에 [9]는 다음

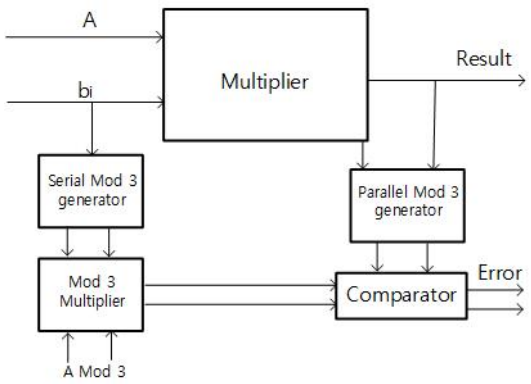

Fig. 1. The Architecture of Error Detection by T.J. Brosnan and N.R. Strader

과 같은 *Mod 3* Adder를 제안하였다.

즉, $S=(s_1,s_0)$, $T=(t_1,t_0)$가 입력이고

$$R=(r_1,r_0)=(S+T)\,Mod3$$

인 경우

$$r_1=(\overline{s_1}\wedge\overline{s_0}\wedge t_1\wedge\overline{t_0})\oplus(s_1\wedge\overline{s_0}\wedge\overline{t_1}\wedge\overline{t_0})\oplus(s_1\wedge\overline{s_0}\wedge t_1\wedge t_0)\oplus(\overline{s_1}\wedge s_0\wedge\overline{t_1}\wedge t_0)\oplus(s_1\wedge s_0\wedge t_1\wedge\overline{t_0}),$$
$$r_0=(\overline{s_1}\wedge\overline{s_0}\wedge\overline{t_1}\wedge t_0)\oplus(s_1\wedge\overline{s_0}\wedge t_1\wedge\overline{t_0})\oplus(\overline{s_1}\wedge s_0\wedge\overline{t_1}\wedge\overline{t_0})\oplus(\overline{s_1}\wedge s_0\wedge t_1\wedge t_0)\oplus(s_1\wedge s_0\wedge\overline{t_1}\wedge t_0)$$

이다. 이는 18개의 *AND gate*와 8개의 *XOR gate*의 공간 복잡도와 $2T_{AND}+3T_{\mathrm{XOR}}$의 *Delay*를 갖는 *Adder*이다.

Fig. 2의 일반적인 *Mod 3 Adder*는 입력이

$$A=(a^H,a^L)=(a_1,a_0)$$

로 표현되어 있다[2,12,13]. 또한 [1]에서는 Fig. 2의 *Adder*에서 입력이 $A_i=(a_1,a_0)$인 경우

$$A_i\,Mod3=((a_1\oplus\mathrm{a_0})\wedge\mathrm{a_1},(\mathrm{a_1}\oplus\mathrm{a_0})\wedge\mathrm{a_0})$$

이므로 (1.1)을 (0,0)으로 변환시키기 때문에 입력단 앞에 이 과정을 추가하는 효율적인 *Mod 3*

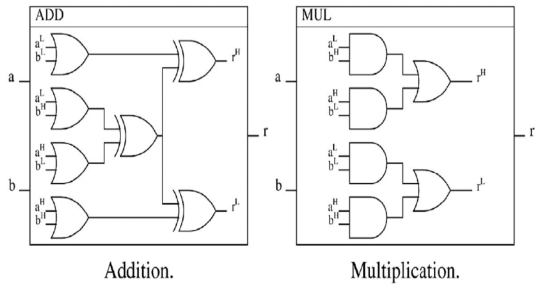

Fig. 2. Adder and Multiplier in *GF(3)*

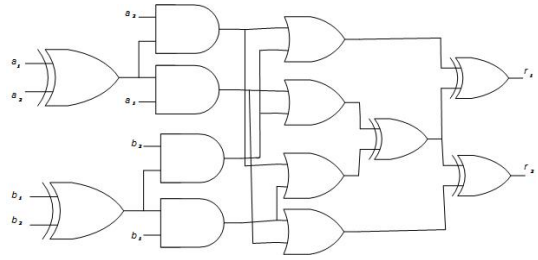

Fig. 3. Mod 3 Adder

*Adder*를 Fig. 3과 같이 제안하였다. 즉, 4개의 OR gate, 5개의 XOR gate, 4개의 AND gate와 $T_{OR}+T_{AND}+3T_{\mathrm{XOR}}$를 시간 복잡도를 갖는다.

III. Modular N 곱셈의 오류 탐지

RSA, ECC 등 대부분 정보보호 관련 분야에 응용되는 *Modular N*의 N은 홀수이다. 본 절 이후로 N을 홀수라 가정하고 $s=2^r-1$이라 하면, 모든 연산은 *Modular N*, 즉 Z_N에서의 연산이다.

Z_N의 두 원소 A, B를

$$A=(\mathrm{A}_{\mathrm{m}-1},\mathrm{A}_{\mathrm{m}-2},\cdots,\mathrm{A}_1,\mathrm{A}_0)_{2^r},$$
$$B=(\mathrm{B}_{\mathrm{m}-1},\mathrm{B}_{\mathrm{m}-2},\cdots,\mathrm{B}_1,\mathrm{B}_0)_{2^r}$$

라 하고, A, B의 연산 결과를

$$C=AB=(C_{2m-2},C_{2m-3},\cdots,C_1,C_0)_{2^r}$$

라 하자.

먼저 A, B를 정수로 보면

$$A\mathrm{Mods}=\sum_{\mathrm{i}=1}^{\mathrm{m}-1}\mathrm{A}_{\mathrm{i}},$$
$$B\mathrm{Mods}=\sum_{\mathrm{i}=1}^{\mathrm{m}-1}\mathrm{B}_{\mathrm{i}}$$

이다. 그리고 AB를 정수로 생각하면 다음과 같다.

$$AB=qN+C,0\le \mathrm{C}<\mathrm{N}\quad .$$

$M=sN$이라 하면

$$AB=q'M+C',0\le \mathrm{C}'<\mathrm{M}\quad ,$$
$$C'=(\mathrm{C}'_{\mathrm{m}},\mathrm{C}'_{\mathrm{m}-1},\cdots,\mathrm{C}'_0)_{2^r}$$

이다. 그러므로

$$AB\mathrm{Mods}=(\sum_{\mathrm{i}=0}^{\mathrm{m}-1}\mathrm{A}_{\mathrm{i}})(\sum_{\mathrm{i}=0}^{\mathrm{m}-1}\mathrm{B}_{\mathrm{i}})\mathrm{Mods}$$
$$=\mathrm{C}'\mathrm{Mods}$$
$$=\sum_{\mathrm{i}=0}^{\mathrm{m}}\mathrm{C}'_{\mathrm{i}}\mathrm{Mods}$$

이다. 위 결과로 다음과 같은 정리를 얻을 수 있다.

정리 1. r은 양의 정수, $s=2^r-1$, N은 홀수, $M=sN$이라 하자. $A,B\in \mathrm{Z_N}$ 에 대하여 Z_M에서 A, B의 곱셈을

$$C'=\mathrm{AB\ \ Mod\ M},$$
$$A=(A_{m-1},Am,\cdots,A_1,A_0)_{2^r},$$
$$B=(B_{m-1},B_{m-2},\cdots,B_1,B_0)_{2^r},$$
$$C'=(\mathrm{C}'_{\mathrm{m}},\mathrm{C}'_{\mathrm{m}-1},\cdots,\mathrm{C}'_0)_{2^r}$$

라 하면

$$(\sum_{i=0}^{m-1}A_i)(\sum_{\mathrm{i}=0}^{\mathrm{m}-1}\mathrm{B}_{\mathrm{i}})\mathrm{Mods}=\sum_{\mathrm{i}=0}^{\mathrm{m}}\mathrm{C}'_{\mathrm{i}}\mathrm{Mods}$$

이다. □

정리 1을 이용하여 Z_N의 곱셈기에 대한 오류 탐지 방법을 Fig. 1과 같이 제시할 수 있다.

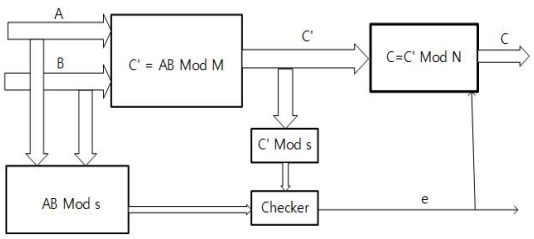

Fig. 4. The Architecture of Error Detection in Z_N

Fig. 4의 모듈러 곱셈의 경우는

$$AB\text{Mods} = (\sum_{i=0}^{m-1} A_i)(\sum_{i=0}^{m-1} B_i)\text{Mods} \quad ,$$

$$C'\text{Mods} = \sum_{i=0}^{m} C'_i\text{Mods}$$

이고, Checker는 Z_s에서의 등호비교 연산이다.

참고. r, s, N, M은 정리 1과 같다 하자. $A,B \in Z_N$ 에 대하여 Z_M에서 A, B의 덧셈을

$$C' = A + B \,\text{Mod}\, M \quad ,$$
$$A = (A_{m-1}, Am, \cdots, A_1, A_0)_{2^r},$$
$$B = (B_{m-1}, B_{m-2}, \cdots, B_1, B_0)_{2^r},$$
$$C' = (C'_m, C'_{m-1}, \cdots, C'_0)_{2^r}$$

라 하면

$$(\sum_{i=0}^{m-1} A_i + \sum_{j=0}^{m-1} B_i)\text{Mods} = \sum_{i=0}^{m} C'_i\text{Mods}$$

이다. 그러므로 모듈러 덧셈 오류도 탐지가 가능하다.

IV. 오류 탐지 곱셈기의 복잡도 분석

$s = 2^r - 1$이므로 N이 m비트인 경우 $M = sN$은 $m+r$비트 이다. 그러므로 직렬과 병렬 모두 Z_M의 연산은 Z_N의 연산 보다 최대 r비트가 추가된다.

4.1 Z_M에서 직렬 곱셈기와 병렬 $Mods$

직렬 곱셈기에서 $Mods$ 를 이용한 오류 탐지를 표현하면 Fig. 5와 같다. 먼저 직렬 $AB\,Mod\,M$ 곱

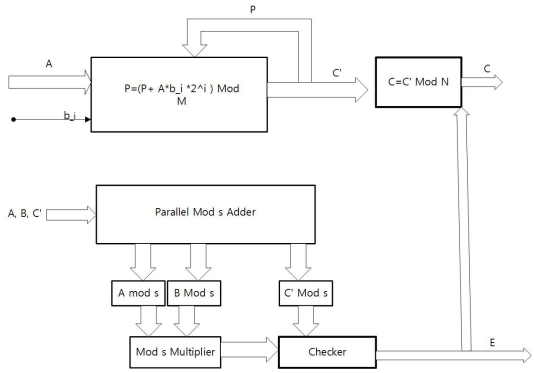

Fig. 5. The Architecture of Error Detection in Serial Multiplier of Z_N

셈기의 특성을 살펴보자[15,16]. 직렬 곱셈기이므로 다음의 연산을 반복한다.

$$(\text{sum} + b_i * (A * 2^i \text{Mod M})) \bmod M$$

구체적으로는 Montgomery 방법 등을 활용할 것이다[15,16]. 이 경우 한번의 i에 필요한 gate를 각 비트별 연산에 AND, $CSA(Carry\ Save\ Adder)$ 또는 $FA(Full\ Adder)$ 2개를 이용한다. 직렬로 $AB\,Mod\,M$ 을 계산하는 동안 $Parallel\ Mods\ Adder$에서는

$$A\text{Mods} = \sum_{i=1}^{m-1} A_i, \quad B\,\text{Mods}$$

를 계산하고 $AB\,Mod\,M$ 의 계산이 끝난 후 C' Mods 를 계산한다. 그리고 $C = C'\ Mod\ N$ 계산은 $2^m = \overline{M} + 1$ 이므로 기존 곱셈기를 이용하여 C' Mods 를 계산하는 동안 계산할 수 있다. 또 $s=3$, 7인 경우는 소프트웨어적으로도 처리가능하다. 그리고 암호학적으로 활용될 경우 모든 계산이 끝난 후 최종적으로 한번 계산하면 된다.

전체적으로 공간 복잡도는 $AB\,Mod\,M$ 에서 N 보다 M이 r비트 증가에 따라 모듈러 곱셈에서 $(r/l)*100\%$ 그리고 $Parallel\ Mods\ Adder$에 $Mods\ \ Adder$ $\lceil (l+r/r) \rceil$ 개, $Mods\ Multiplier$ 1개, r 비트 register 3개, 그리고 $Mods\ \ Checker$ 1개가 필요하다.

r이 2인 $s=3$의 경우는 1988년에 T.J. Brosnan 등[9]이 제시한 것과 같이 $Adder$,

*Multiplier, Checker*를 쉽게 구성할 수 있다. 또한 *Mod 3 Adder*의 경우 T.J. Brosnan 등의 것보다 더 효율적인 C.H Kim[2]의 것을 활용할 수 있다. 그리고 병렬 *Mod* 3 *Adder*의 경우 T.J. Brosnan 등[9]이 제시한 것과 같이 정리 1에서 A_i, B_j, C_k' 가 입력되는 단계는 (1,1)이 입력되는 *Adder*를 쓰고 나머지는 일반 Adder [Fig.2]를 쓰면 더욱 공간복잡도를 줄일 수 있다. 그리고 *Mod*3 곱셈기는 기존의 것[Fig. 2]을 그대로 쓰면 된다. *Mod*3 *Checker*는 $T=(t_1,t_0), S=(s_1,s_0)$ 가 입력인 경우 $\overline{(s_1\oplus t_1)\vee(s_0\oplus t_0)}$이므로 1인 경우 오류가 없고 0인 경우 오류가 있다.

s=7인 경우는 Fig. 6과 같이 *Mod* 7 *Adder*를 제안하였다. *Mod* 7 *Adder*는 9 *AND*, 12 *XOR gate*와 $4T_{\mathrm{XOR}}$의 시간복잡도를 갖는다. 입출력 값으로 (1,1,1)=(0,0,0)도 가능하다. 이 경우가 발생함으로 *Mod* 7 *Checker*는 입력단에서 (a_2,a_1,a_0)가 입력되면

$$((a_0\wedge a_1\wedge a_2)\oplus a_2,(a_0\wedge a_1\wedge a_2)\oplus a_1,\\ (a_0\wedge a_1\wedge a_2)\oplus a_0)$$

로 전환하면 (1,1,1)을 제외한 모든 값은 그대로이고 (1,1,1)은 (0,0,0)으로 변환된다. 이렇게 변환한 후 다음과 같이 *Checker*가 얻어진다.

$$\overline{(s_0\oplus t_0)\vee(s_1\oplus t_1)\vee(s_2\oplus t_2)}$$

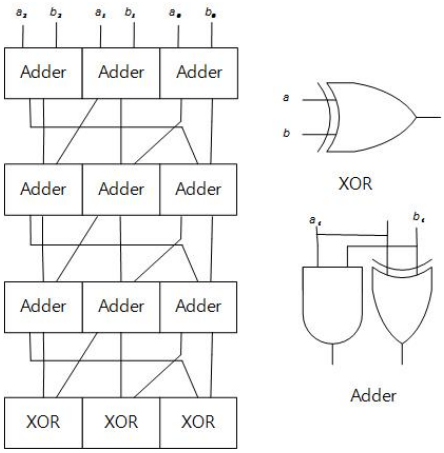

Fig. 6. Mod 7 Adder

결과적으로 전체적인 공간 복잡도는 Table.1과 같다. Table.1에서 기존 *Mod N* 곱셈에서

$$\mathrm{Sum}+b_i*(A*2^i)\mathrm{Mod}N$$

연산 한번에 필요한 *gate*는 각 비트별 연산에 *AND* 2, *FA(Full Adder)* 2개를 기본적으로 사용함으로 *Mod* 3 *adder* 하나의 게이트 수 와 비슷하다. 이런 관계로 *Mod N* 곱셈에서 $\mathrm{Sum}+b_i*(A*2^i)\mathrm{Mod}N$ 연산 한번에 필요한 게이트를 *Mod* 3 *Adder* 1024개로 환산($N\simeq 2^{1024}$)하여 계산하였다. *Mod* 7 *Adder*인 경우는 *Mod* 5의 1.6배로 계산하였다[15,16].

*M*이 *r*비트 증가에 따라 모듈러 곱셈에서 $(r/l)*100\%$증가 값 또한 작아서 0으로 처리하였다. 또한 추가적인 콘트롤에 필요한 게이트도 생략하였다. 시간 복잡도는 $AB\,Mod\,M$ 에서 $AB\,Mod\,N$ 에 비하여 *M*이 *r*비트 증가하지만 추가적인 시간 증가는 없고, C' Mods 에서 $\lceil \log_2((l+r)/r)\rceil$ 번의 *Mods Adder*의 시간, *Mods Checker* 1번의 시간, $C=C'\ Mod\,N$ 을 구하는 시간이 필요하다. 특히, s=3인 경우의 *Mod* 3 *Adder* 한번을 직렬곱셈 한번으로 간주하여 추가적인 시간복잡도를 계산하였으며 구체적인 시간복잡도는 Table.1과 같다. 앞에서 설명했듯이 $C=C'\,Mod\,N$ 에 필요한 추가적인 공간과 시간은 계산하지 않았고, 또한 콘트롤 관련 시간, 공간 복잡도도 제외하였다.

Table 1. Additional Time and Space Complexity

s		Additional Complexity($N\simeq 2^{1024}$)	
		Space Complexity	Time Complexity
3	Additional cost	513 Mod 3 Adder +1 Mod 3 Multiplier +Mod 3 Checker	10 Mod 3 Adder +1 Mod 3 Checker
	%	50%≥	1%≥
7	Additional cost	343 Mod 7 Adder +1 Mod 7 Multiplier +Mod 7 Checker	9 Mod 7 Adder +1 Mod 7 Checker
	%	54%≥	1%≥

4.2 실험결과

*AB Mod M*을 구하는 serial 곱셈기는 다음과 같은 알고리즘으로 계산을 한다.

알고리즘 1

Input : $A=(a_{l-1},\cdots,a_0), B=(b_{l-1},\cdots,b_0), M$

Output : $C=ABModM$

1. C = 0, T=0, k=0
2. While($k < l$)
   - 2.1 T = $A*b_k*2^k ModM$
   - 2.2 C = (C + T) Mod M
   - 2.3 k = k+1
3. Return (C)

오류를 주입하는 시뮬레이션을 위하여 1 비트 오류의 경우 랜덤하게 j, $0 \leq j < l$ 를 선택하여 2.2 또는 2.3에 2^j를 더한 연산 결과로 1 비트 오류를 테스트하였고 2 비트, 3 비트도 똑같이 j를 2, 3개 랜덤하게 선택하여 2.2, 2.3 값에 더하여 계산하는 방법으로 시뮬레이션 하였다. 구체적 오류탐지 시뮬레이션 결과는 Table.2와 같다.

Table 2. Error Detection Probabilities

Number of Error Bit	Error Detection Probabilities(%)	
	s=3	s=7
1 Bit Error	99	99
2 Bit Error	50	99
Random Bit Error	66	82

V. 결 론

RSA를 비롯하여 암호관련 분야 공격에 오류 주입 공격이 강력한 공격 도구로 등장하면서 이를 방지하기 위한 연구가 활발히 진행 되었으나 *Mod N* 오류 탐지는 2014년 제시된 Montgomery 곱셈기를 이용할 경우 오류 탐지 방안이 제시된 것 외에는 특별한 방법이 제시되지 않았다. 본 논문에서는 *Mod s* 를 이용하여 모든 *Mod N* 연산에 대한 오류 탐지를 효율적으로 하는 방법을 제안하였다. 제안한 모듈러 오류 탐지 방법은 $s=3$인 경우 전체적으로 공간 복잡도 50%, 시간복잡도 1% 이하의 추가 부담이 필요하나 1 비트 오류는 99%, 2 비트 오류는 50%가 탐지 가능하다. 특히 $s=7$인 경우는 공간 복잡도는 54%정도 시간 복잡도는 1%정도 증가하나 1,2 비트 오류를 99% 탐지 가능한 강력한 탐지 능력을 갖는다.

References

[1] C.H. Kim, "Efficient Mod 3 Adder for Error Detction," Journal of Institute of Industrial Technology, 23, Semyung University, 2016

[2] N.S. Chang, T.H Kim, C.H. Kim, D.G. Han and H.W. Kim, "Efficient Finite Field Arithmetic Architectures for Pairing Based Cryptosystems," JKIISC, vol. 18, no. 3, pp. 33-43, June. 2008

[3] R.Lidl and H. Niederreiter, "Introduction to finite fields and their applications," Cambridge University Press, Revised Edition, 1994

[4] Nat'l Inst. of Standard and Technology, "Digital Signature Standard," FIPS Publication 182-2, Jan. 2000

[5] IEEE Std. 1363-2000, "IEEE Standard Specification for Public-key Cryptography," Jan. 2000

[6] M. Joye and M. Tunstall, "Fault Analysis in Cryptography," Springer, 2012

[7] D.F. Barbe, "Very Large Scale Integration(VLSI)," New York, Springer-verlag, 1982

[8] N.R Strader and V.T. Rhyne, "A canonical bit-sequential multiplier," IEEE Trans. Comput., Aug. 1982

[9] T. J. Brosnan and N.R. Strader II, " Modular Error Detection for Bit-Serial Multiplication," IEEE Trans. Comput., vol. 37, no. 9, Sep. 1988

[10] G. Zervakis, N. Eftaxiopoulos, K. Tso-umanis, N. Axelos and K. Pekmestzi, "A High Radix Montgomery Multiplier with Concurrent Error Detection," 9th International Design and Test

Symposium, 2014
〔11〕 I. Duursma and H.S. Lee, "Tate pairing implementation for hyper elliptic curves $y^2 = x^p - x + d$," Asiacrypt 2003, LNCS 2894, pp. 111-123, Springer-Verlag, 2003
〔12〕 R. Granger, D. Page, and M. Stam, "Hardware and software normal basis arithmetic for pairing based crypto-graphy in characteristic three," IEEE Transactions on Computers, vol. 54, no. 7, pp. 852-860, July. 2005
〔13〕 D. Page and N. Smart, "Hardware Implementation of Finite Fields of Characteristic Three," CHES 2002, LNCS 2523, pp. 529-539, Springer- Verlag, 2003
〔14〕 C.H. Kim and J-J. Quisqater, "Fault attacks for CRT based RSA : New attacks, New results, and New countermeasures," WiSTP 2007, LNCS 4462, pp. 215-228, 2007
〔15〕 D.N. Amanor, "Efficient Hardware Architectures for Modular Multi-plication," A Thesis of Degree of Master of Science, The University of Applied Science Offenburg, Germany, 2005
〔16〕 J-P. Deschamps, J.L. Imana and G. D. Sutter, "Harhware Implementation of Finite-Field Arithmetic," McGraw Hill, 2009
〔17〕 A. Miyamoto, N. Homma, T. Aoki, and A. Satoh, "Systematic design of RSA processor based on hig-hradix Montgomery Multipliers," IEEE Trans. VLSI Syst. vol. 19, no. 7, pp. 1136-1146, 2011

〈저자소개〉

김 창 한 (Chang Han Kim) 종신회원
1985년 2월: 고려대학교 수학과 이학사
1987년 2월: 고려대학교 수학과 이학석사
1992년 2월: 고려대학교 수학과 이학박사
1992년 8월~현재: 세명대학교 정보통신학부 교수
〈관심분야〉 정수론, 공개키암호, 암호프로토콜

장 남 수 (Nam Su Chang) 종신회원
2002년 2월: 서울 시립대학교 수학과 이학사
2004년 8월: 고려대학교 정보보호대학원 공학석사
2010년 2월: 고려대학교 정보경영공학전문대학원 공학박사
2010년 7월~현재: 세종사이버대학교 정보보호학과 조교수
〈관심분야〉 암호칩 설계 기술, 공개키 암호알고리즘, 공개키암호 암호분석, 부채널 공격