효율적인 NTRU 곱셈기의 하드웨어 구현

김성훈° 오병균 김종화
목포국립대학교 컴퓨터공학
{seilpart°, obk, kimjh}@mokpo.ac.kr

·Hardware Implemetation of Efficient NTRU Multiplier

Seonghoon Kim° Byeongkyun Oh Jonghwa Kim
Dept. of Computer Engineering, Mokpo National University

요 약

임베디드 기기들의 독립적인 통신 접속환경은 공개키 암호화의 요구를 증가시키고 있다. 하지만 이들 기기들의 제한적인 특성(계산능력, 자원, 대역폭)으로 인해 일반적인 공개키 암호화 시스템의 적용이 어렵다. NTRU 공개키 암호화는 이러한 제한점을 극복할 수 있는 성능을 제시한다. 본 논문에서는 이러한 NTRU의 성능을 좀더 향상시키기 위해서 병렬성과 확장성을 고려한 NTRU 곱셈기의 하드웨어 설계를 제안한다.

1. 서론

최근 몇년동안 전통적인 공개키 암호화시스템의 복잡한 계산을 수행할 수 있는 고성능의 퍼스널컴퓨터 보급과 더불어 암호화가 보편화된 웹 브라우저의 발전을 가져왔다. 또한 마이크로프로세서 벤더들에게는 공개키 암호화 기능이 추가된 제품 출하를 요구하였으며, 모듈러 지수화를 한층 가속화한 맞춤형(custom) 구조의 8-bit 마이크로 컨트롤러의 발전을 가져오게 했다.[1]

하지만, 임베디드 시스템의 발전에 의해 세탁기에서 휴대폰, 텔레비전, 오토모빌, 볼루투스 등 여러 기기들이 독립적인 통신이 가능한 상태에 있는데도 불구하고 임베디드 시스템들의 암호화는 초보단계에 있다. 통신이 가능하다는 것은 결국 퍼스널컴퓨터에서와 같이 공개키 암호화와 같은 보안의 필요성이 요구된다.

임베디드 기기들은 일반 데스크탑 장비들과는 달리 제한적인 계산능력을 가지고 있으며, 전력, 메모리, CPU 사이클 등의 요소들이 보완되어야 한다.

본 논문에서는 전통적인 공개키 암호화 시스템보다는 임베디드 시스템들에 적용 가능성을 고려한 NTRU 공개키 암호화 시스템을 소개하고, 병렬성과 확장성을 고려한 NTRU 구현을 위한 NTRU 곱셈기의 하드웨어 설계를 제안하고자 한다.

2. NTRU Public-Key Cryptosystem

NTRU암호시스템은 브라운대학의 수학과 교수인 Hoffstein, Pipher, Silverman이 주축으로 Crypto96에 제안한 새로운 공개키 암호시스템이다. 안전성의 근거는 두개의 서로 다른 module에 의한 다항식 연산의 결과를 분석하는 것은 어렵다는 것이다.

NTRU암호시스템은 암호화 과정에서 랜덤한 원소를 사용하므로 각각의 메시지에 대한 다양한 암호문의 생성이 가능하다. 또 한가지 NTRU암호시스템의 특징은 암호화 및 복호화 시간이 매우 빠르고, 키 생성이 빠르고 쉽다는 것이다.[2]

일반적인 NTRU는 세개의 정수(N, p, q)

- N은 소수
- p and q는 서로소, $gcd(p, q) = 1$
- q 는 p보다 훨씬 크다

로 시작된다. NTRU는 ring $R = Z[x]/(x^N - 1)$ 내의 다항 덧셈과 곱셈을 기반으로 한다. 다항 덧셈이나 곱셈 후 다항식의 계수가 모듈러 q 나 p로 제거된다. 또한 키 생성을 위해 확장 Euclidean 알고리즘을 이용한 두개의 다항 인버스가 요구된다.

2.1 Key Generation

공개키를 위해서

- 모듈로 p로 삭제된 계수를 가진 랜덤한 다항 $f \in R$ 를 비밀키로 선정
- 모듈로 p로 삭제된 계수를 가진 랜덤한 다항 $g \in R$ 을 선정
- 비밀키 f의 모듈러 q에 대한 인버스 다항 F_q를 계산

위의 계산이 완료되면 공개키 h는

$$h = F_q * g \pmod{q}.$$

이 된다.

2.2 Encryption

암호화된 메시지는

$$e = pr * h + m \pmod{q}$$

로 계산되고, 메시지 $m \in R$, 랜덤 다항 $r \in R$은 모두 모듈러 p로 계수가 삭제된다.

2.3 Decryption

복호화 절차는 세가지 스텝을 갖는다.

- $a = f * e \pmod q$
- a의 shift 계수, range $(-\frac{q}{2}, \frac{q}{2})$
- $d = F_p * a \pmod p$

복호화의 마지막 스텝은 모듈러 p로 비밀키 f의 인버스항 F_p를 구한다. 위의 간략한 복호화 절차는 원본 메시지를 구한다. $(d = m)$

3. NTRU Multiplier Design

본 논문에서는 NTRU의 키생성, 암호화, 복호화의 효율을 증가시키는데 있어 중요한 연산인 다항곱셈을 처리하는 곱셈기의 확장가능하고 최적화된 하드웨어 설계를 제안한다.

	a_6	a_5	a_4	a_3	a_2	a_1	a_0
×	b_6	b_5	b_4	b_3	b_2	b_1	b_0
	a_6b_0	a_5b_0	a_4b_0	a_3b_0	a_2b_0	a_1b_0	a_0b_0
	a_5b_1	a_4b_1	a_3b_1	a_2b_1	a_1b_1	a_0b_1	a_6b_1
	a_4b_2	a_3b_2	a_2b_2	a_1b_2	a_0b_2	a_6b_2	a_5b_2
	a_3b_3	a_2b_3	a_1b_3	a_0b_3	a_6b_3	a_5b_3	a_4b_3
	a_2b_4	a_1b_4	a_0b_4	a_6b_4	a_5b_4	a_4b_4	a_3b_4
	a_1b_5	a_0b_5	a_6b_5	a_5b_5	a_4b_5	a_3b_5	a_2b_5
+	a_0b_6	a_6b_6	a_5b_6	a_4b_6	a_3b_6	a_2b_6	a_1b_6
	c_6	c_5	c_4	c_3	c_2	c_1	c_0

그림 1 부분곱 배열, N=7, u=3

3.1 확장성

그림 1과 같이 각각의 부분곱 항들은 모듈러 q로 제거되기 때문에 캐리전달이 열들 사이에는 없고 각 열안에 포함된다. 열들사이의 캐리전달의 부재 때문에 부분곱 어레이 안의 독립된 열들은 병렬로 처리될 수 있다. 부분곱 열을 k에 대해서 하나의 계수 곱셈, 하나의 계수 덧셈, 모듈러 q연산으로 구성된 하나의 프로세싱 유닛은 다음 연산으로 처리된다.

$$c[k]=c[k]+a[i]\cdot b[j] \pmod q \quad j = 0,1,\ldots,N-1$$
$$i = -j \bmod N$$

그러므로, 병렬성은 그림 1의 박스안에 부분곱 열들의 프로세싱 유닛(u)의 수를 조절하므로서 이루어 질 수 있다. 각 프로세싱 유닛은 다른것들과 독립적으로 동작하므로 다중 부분곱 열은 동시에 처리될 수 있다.

3.2 확장가능한 아키텍처

그림 2에 제시한 확장가능한 NTRU 곱셈기는 컨트롤 유닛, 레지스터, 프로세싱 유닛들의 세가지 주요한 컴포넌트로 구성된다.

컨트롤 유닛은 다항곱셈의 계산과 호스트 시스템의 메모리 접근을 제어한다. 곱셈기에는 네 개의 주요한 레지스터가 있다. (두개의 u×8-bit shift register, 하나의 2×2-bit shift register, 하나의 u×8-bit register)

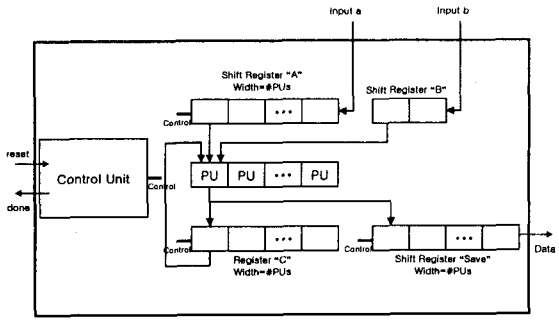

그림 2 NTRU 곱셈기

단일 다항 곱셈을 시행하기 위한 제어 유닛의 전형적인 시퀀스는 그림 3의 알고리즘에 나타난다.

```
 1: if reset = '1' then
 2:   C=0
 3:   Save=0
 4:   x=0 {x is a counter}
 5: else
 6:   A=(a[u-1],…,a[0])
 7:   B=(b[0],b[1])
 8:   for j=0 to [N/u] do
 9:     for i=0 to N-1 do
10:       C=C+(A·B1) mod q {B1은 B의 상위 워드를 표시}
11:       A=(Au-2…A0,a[x-i-1 mod N])
12:       B=(B0,b[i+1 mod N])
13:     end for
14:     Save=C+(A·B1) mod q
15:     C=0
16:     (c[x+u],…,c[x])=Save
17:     x=x+u
18:     A=(a[x+u-1],…,a[x])
19:     B=(B0,b[1])
20:   end for
21: end if
```

그림 3 병렬성이 향상된 NTRU 다항곱셈 알고리즘

좀 더 명확한 설계를 위해 아래를 가정한다.

- 곱셈기는 세 개의 프로세싱 유닛으로 설정(u=3)
- 다항 a와 b, 그리고 c는 각각 7의 계수를 갖는다(N=7)
- 계수 어레이는

$$a(x) = \sum_{i=0}^{N-1} a[i]x^i \leftrightarrow \text{a=(a[N-1],…,a[1],a[0])}$$

- 벡터 A,B,C,Save는 그림 2의 레지스터들이다.
- PU는 부분곱 배열의 u열 $\left\lceil \frac{N+u-1}{u} \right\rceil$세트를 처리.

알고리즘은 초기화단계, 실행단계, 로드 및 작성단계로 분리된다. 초기화단계는 1-7 스텝이고, 실행단계는 9-13, 로드 및 작성단계는 14-19이다.

제어유닛은 표 1에 제시된 초기화단계를 시작한다. 먼저, CC0에 레지스터 C와 Save를 클리어한다. CC1에 레지스터 A의 다항a의 처음 세 개의 (u) 계수와 레지스터 B의 다항b의 처음 두개의 계수를 직렬로 로드한다. CC3에 레지스터 A와 B의 로딩을 완료한다.

cycle	PU2	PU1	PU0
0	reset = '1'		
	C_2=0 $Save_2$=0	C_1=0 $Save_1$=0	C_0=0 $Save_0$=0
1	A_2=0 B_1=0	A_1=0 B_1=0	A_0=a[2] B_1=0
2	A_2=0 B_1=b[0]	A_1=a[2] B_1=b[0]	A_0=a[1] B_1=b[0]
3	A_2=a[2]	A_1=a[1]	A_0=a[0]

표 1 초기화 단계

열의 첫 번째 세트를 위한 실행단계는 표 2에 제시된다. CC4에 프로세싱유닛은 각 입력을 처리하고 제어유닛은 레지스터 C로 각 PU의 출력을 래치한다. 다항a와 b로부터 다음 계수가 각 레지스터에 로드된다. 제어유닛은 CC4에 생성된 연산을 5(N-2)번 이상 반복한다. 마지막으로 CC9에서 이 단계를 위한 모든 연산이 완료된다.

cycle	PU2	PU1	PU0
4	$C_2=C_2+A_2 \cdot B_1$ A_2=a[1] B_1=b[1]	$C_1=C_1+A_1 \cdot B_1$ A_1=a[0] B_1=b[1]	$C_0=C_0+A_0 \cdot B_1$ A_0=a[6] B_1=b[1]
5	$C_2=C_2+A_2 \cdot B_1$ A_2=a[0] B_1=b[2]	$C_1=C_1+A_1 \cdot B_1$ A_1=a[6] B_1=b[2]	$C_0=C_0+A_0 \cdot B_1$ A_0=a[5] B_1=b[2]
6	$C_2=C_2+A_2 \cdot B_1$ A_2=a[6] B_1=b[3]	$C_1=C_1+A_1 \cdot B_1$ A_1=a[5] B_1=b[3]	$C_0=C_0+A_0 \cdot B_1$ A_0=a[4] B_1=b[3]
7	$C_2=C_2+A_2 \cdot B_1$ A_2=a[5] B_1=b[4]	$C_1=C_1+A_1 \cdot B_1$ A_1=a[4] B_1=b[4]	$C_0=C_0+A_0 \cdot B_1$ A_0=a[3] B_1=b[4]
8	$C_2=C_2+A_2 \cdot B_1$ A_2=a[4] B_1=b[5]	$C_1=C_1+A_1 \cdot B_1$ A_1=a[3] B_1=b[5]	$C_0=C_0+A_0 \cdot B_1$ A_0=a[2] B_1=b[5]
9	$C_2=C_2+A_2 \cdot B_1$ A_2=a[3] B_1=b[6]	$C_1=C_1+A_1 \cdot B_1$ A_1=a[2] B_1=b[6]	$C_0=C_0+A_0 \cdot B_1$ A_0=a[1] B_1=b[6]

표 2 실행단계

마지막으로 로드/작성단계는 표 3에 제시한다. CC10에서 PU는 CC9 입력의 마지막 세트를 처리하고 레지스터 Save에 각 PU의 출력을 래치한다. 그리고 레지스터 C를 클리어하며, 그링 1의 두번째 세트 1행의 계수가 쉬프트 레지스터 a에 직렬로 로드되기 시작하고 레지스터 B에 계수가 로드된다.

CC11에서 CC13까지 제어유닛은 외부 Cache에 이전 세트의 최종 결과를 작성한다. 이것은 Save 레지스터($Save_0$)의 최소 유효 슬롯을 외부 Cache로 전송함을 의미한다.

CC12에서 레지스터 A로의 로딩을 마치고, CC13 동안 Save 레지스터의 마지막 계수가 외부 Cache에 작성된다.

제어유닛은 또한 두 번째 열 세트를 위한 실행단계를 시작하고 또한번의 로드/작성 단계의 실행을 한다. 마지막으로 세 번째 컬럼 세트가 처리되고 외부 Cache에 결과가 작성되며 곱셈이 완료된다.

cycle	PU2	PU1	PU0
10	$Save_2=C_2+A_2 \cdot B_1$ C_2=0 A_2=a[2] B_1=b[0]	$Save_1=C_1+A_1 \cdot B_1$ C_1=0 A_1=a[1] B_1=b[0]	$Save_0=C_0+A_0 \cdot B_1$ C_0=0 A_0=a[5] B_1=b[0]
11	c_2 = $Save_0$ Save = (0, $Save_2$…$Save_1$)		
	A_2=a[1]	A_1=a[5]	A_0=a[4]
12	c_1 = $Save_0$ Save = (0, $Save_2$…$Save_1$)		
	A_2=a[5]	A_1=a[4]	A_0=a[3]
13	c_0 = $Save_0$		

표 3 로드/작성 단계

4. 결론 및 추후 연구

본 논문에서 제안한 NTRU 곱셈기는 다항곱셈의 병렬적인 처리가 가능하므로 최적화되고 확장가능한 NTRU시스템의 하드웨어 설계를 가능하게 할 것이다. 차후 VHDL을 이용하여 FPGA와 같은 유연성이 많은 디바이스로 구현 검증한다면 임베디드 기기에 적합한 통합된 디자인의 임베디드 보안 솔루션을 제공할 수 있을 것이다.

참고문헌

[1] D.Bailey, D. Coffin, A.Elbrit, J. Silverman, and Woodbury, "NTRU in Constrained Devices", in Workshop on Cryptographic Hardware and Embedded Systems, pp. 5-6, May. 2001

[2] J.Hoffstein, J.Pipher, and J.Siverman. NTRU : A new high speed public key cryptosystem. In J. Buhler, editor, Lecture Notes in Computer Science1423 : Algorithmic Number Theory(ANTS III), pp. 267-288, Springer-Verlag, Berlin 1998.

[3] J. Hoffstein and J. H. Silverman, "Optimizations for NTRU", in Proceedings of Public Key Cryptography and Computational Number Theory, de Gruyter, Warsaw, September 2000