Journal of The Korea Institute of Information Security & Cryptology
VOL.27, NO.5, Oct. 2017

ISSN 1598-3986(Print)
ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2017.27.5.1001

Ring-LWE 기반 공개키 암호시스템의 선택 암호문 단순전력분석 공격 대응법*

박 애 선,[1†] 원 유 승,[1] 한 동 국[1,2‡]
[1]국민대학교 금융정보보안학과, [2]국민대학교 정보보안암호수학과

Countermeasure against Chosen Ciphertext Spa Attack of the Public-Key Cryptosystem Based on Ring-Lwe Problem*

Aesun Park,[1†] Yoo-Seung Won,[1] Dong-Guk Han[1,2‡]
[1]Dept. of Financial Information Security, Kookmin University,
[2]Dept. of Information Security, Cryptology, and Mathematics, Kookmin University

요 약

격자 기반 암호는 양자 컴퓨터 공격에 대응 가능한 포스트 양자 암호 중 하나로 알려져 있다. 그 중 ring-LWE 문제는 LWE의 대수적 변종으로 벡터 대신 환(ring)의 원소를 이용한다. 포스트 양자 암호라 할지라도 실제 디바이스에 이를 적용 할 때 부채널 분석 취약점이 존재한다는 것은 이미 알려져 있다. 실제 2016년 Park 등은 Roy 등이 제안한 NTT를 이용한 ring-LWE 기반 공개키 암호시스템의 SPA 취약점을 보고했으며, Reparaz 등은 Roy 암호에 대한 DPA 공격 및 대응법을 2015년과 2016년에 제안하였다. 본 논문에서는 Roy 암호에 대하여 Park 등이 제안한 선택 암호문 SPA 공격이 NTT를 적용하지 않은 Lyubashevsky 암호의 경우에도 동일하게 적용 가능함을 보인다. 또한 선택 암호문 SPA 공격에 안전한 대응기법을 제안하고 실험적으로 안전성을 검증한다.

ABSTRACT

A lattice-based cryptography is known as one of the post-quantum cryptographies. Ring-LWE problem is an algebraic variant of LWE, which operates over elements of polynomial rings instead of vectors. It is already known that post-quantum cryptography has side-channel analysis vulnerability. In 2016, Park et al. reported a SPA vulnerability of the public key cryptosystem, which is proposed by Roy et al., based on the ring-LWE problem. In 2015 and 2016, Reparaz et al. proposed DPA attack and countermeasures against Roy cryptosystem. In this paper, we show that the chosen ciphertext SPA attack is also possible for Lyubashevsky cryptosystem which does not use NTT. And then we propose a countermeasure against CCSPA(Chosen Ciphertext SPA) attack and we also show through experiment that our proposed countermeasure is secure.

Keywords: Ring-LWE cryptosystem, Simple Power Analysis, countermeasure, lattice-based cryptography

I. 서 론

RSA, Diffie-Hellman(DH) 등과 같은 공개키 암호 시스템은 소인수분해 문제 또는 이산 로그 문제의 어려움에 기반 한다. 그러나 양자 컴퓨터 및 Shor의 알고리즘을 이용하면 소인수분해 및 이산 로

Received(06. 29. 2017), Modified(08. 08. 2017), Accepted(08. 11. 2017)
* 이 논문은 2017년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No. 20170005200011001, (ICT 기초연구실) SCR-Friendly 대칭키 암호 및 응용 코드 개발)
† 주저자, aesons@kookmin.ac.kr
‡ 교신저자, christa@kookmin.ac.kr(Corresponding author)

그 문제는 다항식 시간 내에 풀려 현재 널리 사용되는 암호를 무력화 시키는 현실적인 위협으로 다가올 수 있다[1]. 양자 컴퓨터의 발전에 따라, 우리는 차츰 현재 공개키 암호 시스템을 양자 컴퓨터 공격에 대응 가능한 새로운 공개키 암호 시스템으로 대체해야 한다.

격자 기반 암호는 양자 컴퓨터 공격에 대응 가능한 포스트 양자 암호 중 하나로 알려져 있다. 격자는 기저벡터 집합의 정수 계수로 표현할 수 있는 점들의 그룹으로 기저벡터에 의해 충분히 정의되었다 할지라도 동일한 격자를 표현 할 수 있는 무수히 많은 수의 다른 기저벡터가 존재한다. 이러한 특징으로 인해 임의의 기저벡터로 구성된 격자가 주어졌을 때 격자의 벡터 중 0이 아닌 가장 짧은 벡터를 찾는 SVP(Shortest Vector Problem), 임의의 벡터 v가 주어졌을 때 v와 가장 가까운 격자 벡터를 구하는 CVP(Closest Vector Problem)와 노이즈가 추가된 선형 방정식으로부터 생성된 벡터를 균등한 랜덤 벡터로부터 구별하는 LWE(Learning With Error) 등의 어려운 문제가 존재한다. LWE는 암호학적 프리미티브(primitive)의 구성에 사용되는 다양한 문제로 사용된다. 그러나 LWE를 사용하면 키 크기가 커지는 단점이 있다. 한편, ring-LWE 문제는 LWE의 대수적 변종으로 벡터 대신 환(ring)의 원소를 이용한다.

이론적으로 안전한 암호 알고리즘이라도 실제 디바이스에서 동작 될 때 소비 전력 등 부가적인 정보를 노출한다. 이러한 부가적인 정보를 이용한 공격을 부채널 분석이라 하며 1996년 Kocher에 의해 제안된 이후, 많은 암호 알고리즘에 대한 부채널 분석 연구가 이루어져왔다[2].

포스트 양자 암호 알고리즘이라 할지라도 실제 디바이스에 적용될 때 부채널 분석 취약점이 존재한다. 실제 포스트 양자 암호에 대한 부채널 공격 및 대응법이 10년 전부터 연구되어 왔다[3-7]. 2010년 Lyubashevsky 등[8]이 ring-LWE 기반 공개키 암호시스템을 제안한 이후로 2014년 ring-LWE 암호시스템의 효과적인 구현을 위해 NTT(Number Theoretic Transform)를 이용한 방법이 Roy 등에 의해 제안되었다[9]. 이후 NTT를 적용한 ring-LWE 기반 공개키 암호시스템에 대한 부채널 분석 및 대응법 연구가 2015년 이후로 계속 이루어지고 있다.

본 논문에서는 NTT를 적용한 ring-LWE 기반 공개키 암호시스템에 대한 Park 등[7]이 제안한 선택 암호문 단순전력분석(Chosen Ciphertext Simple Power Analysis, CCSPA) 공격이 NTT를 적용하지 않을 때에도 동일한 취약점이 존재함을 보인다. 또한 CCSPA 공격을 방지하기 위한 대응법을 제안한다.

본 논문의 구성은 2장에서 ring-LWE 암호시스템 및 Park 등이 제안한 선택 암호문 SPA 공격에 대해 설명 후, 3장에서 제안하는 공격 방법 및 대응법을 서술한다. 4장에서는 대응법이 적용된 모듈러 덧셈이 모듈러 연산 여부에 상관없이 항상 동일한 전력 파형을 생성함을 실험을 통해 보이고, 각 알고리즘에 대한 성능 비교 후 결론을 맺는다.

II. 사전 지식

2.1 Ring-LWE 기반 공개키 암호시스템

Lyubashevsky 등에 의해 제안된 ring-LWE 기반 공개키 암호시스템은 기존의 공개키 시스템과 같이 키 생성, 암호화 그리고 복호화 세 단계로 구분된다[8]. 본 절에서는 Lyubashevsky 등이 제안한 키 생성 및 암·복호화 방법과 Roy 등이 제안한 NTT를 이용한 방법에 대해 간단히 언급한다. 이후 우리는 각각의 방법을 Lyubashevsky 암호 및 Roy 암호라 명명한다.

2.1.1 기호 정의

암호시스템 설명에 앞서 본 논문에서 사용되는 기호에 대해 설명한다. 영문 소문자 r_i, c_i, a는 다항식을 의미한다. 특히, a는 공개 가능한 랜덤 다항식으로 공개키에 사용된다. 소문자 m은 메시지로 비트 문자열을 의미하며 q는 정수이다. 다항식은 다음의 식과 같이 표현된다.

$$r(x) = \sum_{j=0}^{n-1} r[j]x^j = r[0]x^0 + r[1]x^1 + \cdots + r[n-1]x^{n-1}$$

여기에서 $r[j]$는 $r(x)$의 x^j의 계수를 의미하며 Z_q의 원소이다. 또한 본 논문에서 사용하는 다항식은 환 $R_q = Z_q[x]/\langle f(x)\rangle$의 원소로 사용되는 n차 기약 다항식은 다음과 같다.

$f(x)=x^n+1$, 단, n은 2의 거듭제곱 꼴

또한 기호 $*$는 두 다항식의 곱을 의미하며, $\bullet$는 두 다항식의 계수별 곱을 의미한다. $\times$는 상수 곱을 나타낸다. 틸더($\sim$) 표시의 영문 소문자는($\tilde{r}$) 다항식(r)의 NTT 적용 결과 값이다.

Ring-LWE 기반 암호 알고리즘을 사용 할 때 파라미터 (n,q,σ)는 일반적으로 $\left(256, 7681, \frac{11.31}{\sqrt{2\pi}}\right)$ 또는 $\left(512, 12289, \frac{12.18}{\sqrt{2\pi}}\right)$로 사용하는데, 이는 각각 128비트 또는 256비트 보안 레벨을 만족하는 파라미터이다. 여기에서 n은 다항식 환의 차수를 의미하고, q는 모듈러스(modulus), σ는 오류 다항식을 선택하는 가우시안 분포의 표준 편차를 의미한다 [10].

2.1.2 Lyubashevsky 암호

1. $KeyGen(a)$: 두 개의 오류 다항식 r_1, r_2를 이산 정규 분포에서 랜덤하게 추출 후, 공개키로 사용되는 다항식 $p=r_1-a*r_2$를 계산한다. (본 논문에서 언급하는 이산 정규 분포는 평균 0과 표준편차 σ를 따른다.) 개인키는 r_2이며, 다항식 쌍 (a,p)를 공개키로 사용한다.

2. $Enc((a,p),m)$: n비트 문자열 메시지 m의 암호화 첫 번째 단계는 메시지 m을 R_q의 원소 $(=\overline{m})$로 인코딩하는 것이다. 인코딩 방법은 메시지 각 비트에 $\left\lfloor \frac{q}{2} \right\rfloor$ 을 곱하여 이루어진다. 여기에서 $\lfloor x \rfloor$는 x를 넘지 않는 최대 정수를 의미한다. R_q의 원소로 인코딩 된 메시지는 아래의 과정을 거쳐 암호문 (c_1,c_2)로 계산된다.
이산 정규 분포에서 오류 다항식 e_1, e_2, e_3를 랜덤하게 추출 후, 아래의 식으로 계산한다.

$$c_1=a*e_1+e_2\ ,\ c_2=p*e_1+e_3+\overline{m}$$

3. $Dec((c_1,c_2),r_2)$: 개인키 r_2를 이용하여 메시지를 복호화 단계로 아래의 식으로 계산된다.

$$m=Decode(c_1*r_2+c_2)$$

여기에서 $Decode$ 함수는 입력 다항식의 각 계수에 대하여 각각 계산되어 지며 아래와 같이 정의한다.

$$Decode(a(x))$$
$$=De'(a[n-1]) \parallel De'(a[n-2]) \parallel \cdots \parallel De'(a[0])$$

$$De'(a[i])=\begin{cases} 0 \text{ if } a[i]\in\left\{\left(0,\frac{q}{4}\right)\cup\left(\frac{3q}{4},q\right)\right\} \\ 1 \text{ if } a[i]\in\left(\frac{q}{4},\frac{3q}{4}\right) \end{cases}$$

2.1.3 Roy 암호

앞서 언급한바와 같이, Roy 암호는 Lyubashevsky 암호의 효과적인 구현을 위해 NTT를 적용한 것으로 NTT 도메인으로 넘겨진 모든 다항식의 곱은 계수별 곱으로 이루어진다는 특징이 있다.

1. $KeyGen(a)$: 두 개의 오류 다항식 r_1, r_2를 이산 정규 분포에서 랜덤하게 추출 후 a, r_1, r_2에 NTT 알고리즘을 적용한다.

$$\tilde{r_1}=NTT(r_1)\ ,\ \tilde{r_2}=NTT(r_2)\ ,\ \tilde{a}=NTT(a)$$

이후 공개키로 사용될 다항식 $\tilde{p}=\tilde{r_1}-\tilde{a}\bullet\tilde{r_2}$를 계산한다. 개인키는 $\tilde{r_2}$이며, 다항식 쌍 $(\tilde{a},\tilde{p})$를 공개키로 사용한다.

2. $Enc((\tilde{a},\tilde{p}),m)$: 메시지 m을 Lyubashevsky 암호와 동일한 방법으로 R_q의 원소 $(=\overline{m})$로 인코딩 후, 암호문 $(\tilde{c_1},\tilde{c_2})$를 다음의 방법을 이용해 계산한다.
이산 정규 분포에서 오류 다항식 e_1, e_2, e_3를 추출 후, e_1, e_2에 NTT 알고리즘을 적용한다.

$$\tilde{e_1}=NTT(e_1)\ ,\ \tilde{e_2}=NTT(e_2)$$

이후 e_3, 공개키 $(\tilde{a},\tilde{p})$ 및 인코딩된 메시지 $\overline{m}$을 이용하여 암호문 $(\tilde{c_1},\tilde{c_2})$을 아래와 같이 계산한다.

$$\tilde{c_1} = \tilde{a} \bullet \tilde{e_1} + \tilde{e_2},\ \tilde{c_2} = \tilde{p} \bullet \tilde{e_1} + NTT(e_3 + \overline{m})$$

3. $Dec((\tilde{c_1}, \tilde{c_2}), \tilde{r_2})$: 개인키 $\tilde{r_2}$를 이용한 메시지 복호화 단계로 아래의 식으로 계산된다.

$$m = Decode(INTT(\tilde{c_1} \bullet \tilde{r_2} + \tilde{c_2}))$$

여기에서 INTT는 역 NTT를 의미한다.

2.2 선택 암호문 단순전력분석

본 절에서는 Park 등이 제안한 CCSPA에 대해 간략히 소개한다. 이 공격은 Roy 암호가 8 비트 마이크로 컨트롤러에서 동작할 때 발생하는 취약점을 이용한다. 앞서 언급한 것과 같이 Roy 암호는 NTT를 적용하여 모든 다항식의 곱은 같은 차수의 계수 곱으로 계산된다.

Algorithm 1.은 복호화 단계를 간략히 나타낸 알고리즘이고, Algorithm 2.는 Algorithm 1.의 1단계 및 2단계를 실제 구현 할 때의 흐름을 의사코드로 나타낸 알고리즘이다. 제안된 공격 기법은 Algorithm 2.의 9~11단계의 조건문 수행 여부를 활용하여 수행되는 것으로, 두 다항식의 덧셈에서 계수별로 덧셈이 이루어질 때 덧셈의 결과 값에 대한 모듈러 연산 발생 여부를 시간(timing) 기반 단순전력분석을 통해 구별가능하다는 취약점을 이용한다.

CCSPA 공격은 각 공격에 암호문을 다르게 선택하여 비밀키를 복원한다. Algorithm 3.은 CCSPA의 흐름을 보여주고 있다. 앞서 언급 한 것 같이, 공격자는 비밀키 $\tilde{r_2}$를 복원하기 위해 각 시행에 필요한 암호문을 다르게 선택해야 한다.

Algorithm 3.의 1~3단계는 첫 번째 시행에 필요한 암호문을 설정하는 단계로, $\tilde{c_1}$, $\tilde{c_2}$의 모든 계수를 1, $\frac{q-1}{2}$로 각각 고정한다. Roy 암호는 NTT를 적용한 다항식의 곱이 이루어지기 때문에 각 시행에 사용되는 $\tilde{c_1}$의 모든 n개의 계수를 항상 1로 고정한다. Algorithm 3.의 8~12단계는 2번째 시행 및 $\lceil \log_2 q \rceil - 1$번째 시행에 사용되는 암호문 $\tilde{c_2}$를 설정하는 단계로 이전 시행의 모듈러 연산 발생 여부에 따라 암호문을 선택한다. $\tilde{r_2}$를 복원하기 위한 마

Algorithm 1 Decryption of Roy cryptosystem

Input: Ciphertext $(\tilde{c_1}, \tilde{c_2})$, Secret Key $\tilde{r_2}$
Output: Encoded message m

1: $\tilde{c_1} \leftarrow$ Coefficient-wise Mul $(\tilde{c_1}, \tilde{r_2})$
2: $\tilde{c_1} \leftarrow$ Coefficient-wise Add $(\tilde{c_1}, \tilde{c_2})$
3: $\bar{m} \leftarrow$ INTT $(\tilde{c_1})$
4: $m \leftarrow$ Decode $(\bar{m})$
5: **return** m

Algorithm 2 Detail operation in Algorithm 1.

Input: Ciphertext $(\tilde{c_1}, \tilde{c_2})$, Secret Key $\tilde{r_2}$
Output: Encoded message $\bar{m}$

1: **for** $i = 0$ to $n-1$ **do**
2: $\quad \tilde{c_1}[i] \leftarrow \tilde{r_2}[i] \cdot \tilde{c_1}[i]$
3: $\quad$ **if** $\tilde{c_1}[i] \geqslant q$ **then**
4: $\quad\quad \tilde{c_1}[i] \leftarrow \tilde{c_1}[i] \mod(q)$
5: $\quad$ **end if**
6: **end for**
7: **for** $i = 0$ to $n-1$ **do**
8: $\quad \tilde{c_1}[i] \leftarrow \tilde{c_1}[i] + \tilde{c_2}[i]$
9: $\quad$ **if** $\tilde{c_1}[i] \geqslant q$ **then**
10: $\quad\quad \tilde{c_1}[i] \leftarrow \tilde{c_1}[i] \mod(q)$
11: $\quad$ **end if**
12: **end for**
13: $\bar{m} \leftarrow$ INTT $(\tilde{c_1})$
14: $m \leftarrow$ Decode $(\bar{m})$
15: **return** m

Algorithm 3 Chosen Ciphertext SPA (CCSPA)

Input: q, n
Output: $\tilde{r_2}[i]$ where $i \in [0, n-1]$

1: **for** $i = 0$ to $n-1$ **do**
2: $\quad \tilde{c_1}[i] \leftarrow 1$, $\tilde{c_2^1}[i] \leftarrow \frac{q-1}{2}$, $k[i] \leftarrow 1$
3: **end for**
4: **Check** the occurrence of the reduction operation
5: **for** $t = 2$ to $\lceil \log_2 q \rceil$ **do**
6: $\quad$ **if** $t \neq \lceil \log_2 q \rceil$ **then**
7: $\quad\quad$ **for** $i = 0$ to $n-1$ **do**
8: $\quad\quad\quad$ **if** modular reduction operation didn't occur **then**
9: $\quad\quad\quad\quad k[i] \leftarrow 2 \times k[i] + 1$, $\tilde{c_2^t}[i] \leftarrow \lceil \frac{q-1}{2^t} \times k[i] \rceil$
10: $\quad\quad\quad$ **else**
11: $\quad\quad\quad\quad k[i] \leftarrow 2 \times k[i] - 1$, $\tilde{c_2^t}[i] \leftarrow \lceil \frac{q-1}{2^t} \times k[i] \rceil$
12: $\quad\quad\quad$ **end if**
13: $\quad\quad$ **end for**
14: $\quad\quad$ **Check** the occurrence of the reduction operation
15: $\quad$ **else**
16: $\quad\quad$ **for** $i = 0$ to $n-1$ **do**
17: $\quad\quad\quad$ **if** modular reduction operation didn't occur **then**
18: $\quad\quad\quad\quad \tilde{c_2}^{\lceil \log_2 q \rceil}[i] \leftarrow \tilde{c_2}^{\lceil \log_2 q \rceil - 1}[i] + 1$
19: $\quad\quad\quad$ **else**
20: $\quad\quad\quad\quad \tilde{c_2}^{\lceil \log_2 q \rceil}[i] \leftarrow \tilde{c_2}^{\lceil \log_2 q \rceil - 1}[i] - 1$
21: $\quad\quad\quad$ **end if**
22: $\quad\quad\quad$ **Check** the occurrence of the reduction operation
23: $\quad\quad\quad$ **if** modular reduction operation didn't occur **then**
24: $\quad\quad\quad\quad \tilde{r_2}[i] \leftarrow q - \tilde{c_2}^{\lceil \log_2 q \rceil}[i] - 1$
25: $\quad\quad\quad$ **else**
26: $\quad\quad\quad\quad \tilde{r_2}[i] \leftarrow q - \tilde{c_2}^{\lceil \log_2 q \rceil}[i]$
27: $\quad\quad\quad$ **end if**
28: $\quad\quad$ **end for**
29: $\quad$ **end if**
30: **end for**

지막 시행의 암호문 $\tilde{c_2}$는 Algorithm 3.의 17~21단계의 값으로 설정한다.

여기에서 $c_j^t[i]$는 t번째 복호화 시행에서의 암호문 c_j의 i번째 계수를 의미하며, $k[i]$는 $c_j^t[i]$를 선택하기 위한 상수이다. $\lceil x \rceil$은 천장 함수(ceiling function)로 x보다 작지 않은 최소 정수를 의미한다.

한 번의 복호화 시행은 비밀키의 후보를 약 $\frac{1}{2}$배 감소시키기 때문에 공격자는 오직 $\lceil \log_2 q \rceil$번의 복호화 시행으로 비밀키를 알아낼 수 있다. 앞서 언급했듯이 q는 보안 레벨과 관련된 파라미터로 128비트 또는 256비트의 보안 레벨을 만족하기 위해 각각 7681, 12289를 사용한다.

III. 제안하는 공격 및 대응법

3.1 Lyubashevsky 암호에 대한 CCSPA

기존의 ring-LWE 기반 공개키 암호시스템에 대한 부채널 분석 및 대응법 연구는 Roy 암호에 대해서만 진행되어 왔다. 본 논문에서는 간단한 아이디어를 통해 Park 등이 제안한 공격이 Lyubashevsky 암호에도 적용 가능함을 보인다.

Roy 암호와 달리 Lyubashevsky 암호에서 a, b가 R_q의 원소일 때, 다항식 a와 b의 곱 $c(c(x) = a(x) * b(x))$ 는 계수 곱으로 이루어지지 않고 컨볼루션(convolution) 곱으로 다음 식과 같이 표현할 수 있다.

$$c[i] = \sum_{j=0}^{i} (a[j]b[i-j] \bmod q) - \sum_{j=i+1}^{n-1} (a[j]b[n+i-j] \bmod q) \quad (\because x^n \equiv -1 \bmod (x^n+1))$$

또한 Lyubashevsky 암호라 할지라도 두 다항식의 덧셈은 계수별로 계산되어 지기 때문에 암호문 c_1을 잘 선택한다면 Park 등이 제안한 동일한 공격이 Lyubashevsky 암호에서도 가능하다.

아이디어는 다음과 같다. CCSPA를 수행하기 위해서는 복호화 연산의 첫 번째 단계인 두 다항식의 곱의 결과를 비밀키 r_2와 동일하게 구성하도록 암호문을 선택하면 된다. 따라서 모든 공격에 $c_1(x) = 1$로 선택한다. 즉,

$$c_1[i] = 0, \ ^{\forall} i \in \{1,2,\cdots n-1\}, \ c_1[0] = 1.$$

이후 c_2의 선택은 Park 등이 제시한 방법을 사용하여 비밀키 복구가 가능하다.

3.2 CCSPA에 안전한 모듈러 덧셈

CCSPA 공격은 모듈러 덧셈 연산이 진행 될 때 조건문의 수행여부가 시간(timing) 기반 단순전력분석을 통해 구별가능하다는 취약점을 이용하는 것이다. 따라서 본 논문에서는 공격을 방지하기 위해 동일 시간에 동작하는 CCSPA에 안전한 모듈러 덧셈을 제안한다.

3.2.1 기호 정의

모듈러 덧셈에 사용되는 모듈러스(modulus) q는 128, 256비트 보안 레벨을 만족하기 위해 각각 7681, 12289를 사용한다. 따라서 $0 \le X \le q-1$ 값은 13비트 또는 14비트로 표현되므로 8비트 마이크로컨트롤러에서 알고리즘이 동작할 때 2개의 레지스터가 필요하기에 아래와 같이 덧셈의 결과를 정의한다.

$0 \le A, B \le q-1$을 만족하는 두 상수 A,B의 일반 덧셈($Z = A+B$) 또는 모듈러 덧셈($Z = A+B \bmod q$) 결과를 다음과 같이 표현한다.

$$Z = Z[1] \parallel Z[0] = z_{15} z_{14} z_{13} \cdots z_2 z_1 z_0$$

여기에서 $Z[i] \in \{0,1\}^8 \ ^{\forall} i \in \{0,1\}$, $z_j \in \{0,1\} \ ^{\forall} j \in \{0,\cdots,15\}$를 의미한다.

3.2.2 CCSPA에 안전한 모듈러 덧셈

$0 \le A, B \le q-1$을 만족하는 두 상수 A,B의 모듈러 덧셈 시행을 위한 처음 단계는 $Z = A+B$를 계산하는 것이다. 이후 계산된 결과가 q 보다 클 경우 Z에서 q를 한번 빼는 단계를 거치게 되는데, 이때 공격자가 이를 구별 할 수 있다면 CCSPA 공격을 이용해 비밀키를 알아 낼 수 있다. 따라서

CCSPA에 안전한 모듈러 덧셈을 위한 가장 기본 적인 아이디어는 처음 단계에서 계산된 Z의 모듈러 연산 부분을 Z값에 상관없이 항상 동일한 연산이 일어나도록 하는 것이다.

본 논문에서는 CCSPA에 안전한 모듈러 덧셈을 위해 처음 단계에서 계산된 Z를 이용하여 $Zflag$와 Z_0flag 계산 후 모듈러 연산에 활용한다. 여기에서 $Zflag$는 Z의 모든 값(즉, $Z[1] \| Z[0]$)을 이용하여 계산되는 값으로 모듈러 연산이 필요한 경우($Z \geq q$) 1, 모듈러 연산이 필요하지 않는 경우($0 \leq Z < q$) 0의 값을 갖도록 구성한다. 이와 비슷하게 Z_0flag는 $Z[0]$를 사용하여 계산되는 값으로 $Z[0]=0$인 경우 0, 그 외의 경우에 1을 갖도록 구성한다.

앞서 언급한 바와 같이 모듈러스 q는 $7681(=0x1E01)$ 또는 $12289(=0x3001)$을 사용하므로, 만약 처음 단계 출력 값 Z가 q보다 크거나 같고 $Z[0]=0$이라면 새롭게 업데이트 될 $Z[1]$은 q의 상위 바이트 $(q \wedge 0xff00)$와 함께 받아내림(borrow)으로 인해 1을 빼야한다. 즉 모듈러 연산이 일어 날 때 새로 업데이트 되는 $Z = Z[1] \| Z[0]$는 처음 단계 출력 값 Z를 이용하여 아래와 같이 표현 할 수 있다.

$$Z[1] \leftarrow \begin{cases} Z[1]-(q \wedge 0xff00)-1 & \text{if } Z[0]=0 \\ Z[1]-(q \wedge 0xff00) & otherwise \end{cases}$$
$$Z[0] \leftarrow Z[0]-(q \wedge 0xff)$$

각 $Zflag$ 및 Z_0flag에 따라 업데이트 되는 값을 다음과 같이 3가지 경우로 나눌 수 있다.

case 1) $Zflag=0$

$Zflag=0$이라는 것은 모듈러 연산이 발생하지 않는 것으로 처음 단계 출력 값이 그대로 할당된다.

$$Z[1] \leftarrow Z[1],\ Z[0] \leftarrow Z[0]$$

case 2) $Zflag=1, Z_0flag=0$

앞서 언급한 것과 같이 $Zflag=1$이고 $Z_0flag=0$이라는 것은 $Z[0]=0$이면서 모듈러 연산이 발생하는 것을 의미하므로 아래의 식으로 적용된다.

$$Z[1] \leftarrow Z[1]-(q \wedge 0xff00)-1$$
$$Z[0] \leftarrow Z[0]-(q \wedge 0xff)$$

case 3) $Zflag=1, Z_0flag=1$

마지막으로 $Zflag=1$이고 $Z_0flag=1$인 경우는 다음과 같이 할당된다.

$$Z[1] \leftarrow Z[1]-(q \wedge 0xff00)$$
$$Z[0] \leftarrow Z[0]-(q \wedge 0xff)$$

$Zflag$와 Z_0flag 계산은 다음과 같이 이루어진다. 예를 들어 $q=7681$인 경우 처음 단계에서 계산된 Z의 값은 Fig.1.과 같이 표현 할 수 있다. 덧셈의 결과가 $7680 \leq Z \leq 8291$인 경우, 모든 Z에 대하여 $z_{12}=z_{11}=z_{10}=z_9=1$로 할당되어 있다. 그 중 모듈러 연산이 일어나지 않는 $Z=7680$인 경우에 대해서만 $z_j=0, {}^{\forall}j \in \{0,\cdots,8\}$이고 이외의 다른 Z는 $z_j=1,\ j \in \{0,\cdots,8\}$인 비트가 적어도 하나 존재한다. 이러한 특징을 이용하여 $7681 \leq Z \leq 8291$ 모듈러 연산이 발생하는 경우에 대해 $Zflag=1$이 설정되도록 아래의 식을 이용한다.

$$(z_{12} \wedge z_{11} \wedge z_{10} \wedge z_9 \wedge (z_8 \vee z_7 \vee \cdots \vee z_1 \vee z_0))$$

또한 $8292 \leq Z \leq 15360$인 경우에는 항상 $z_{13}=1$이므로 이 경우에 대해서는 z_{13}의 비트만 확인 하면 된다. 따라서 모든 덧셈 결과 Z에 대하여 $Zflag$ 값은 다음의 식을 이용한다.

$$z_{13} \vee (z_{12} \wedge z_{11} \wedge z_{10} \wedge z_9 \wedge (z_8 \vee z_7 \vee \cdots \vee z_1 \vee z_0))$$

이때 Z_0flag는 $Z[0]=0$인 경우에 0, 다른 경우에는 1을 갖도록 아래의 식을 이용한다.

$$Z_0flag = z_7 \vee \cdots \vee z_1 \vee z_0$$

256비트 보안 레벨을 만족하기 위해 사용되는 모듈러스(=12289)에 대하여도 유사한 방법이 적용되어 $Zflag$ 값은 다음의 식을 이용한다.

$$z_{14} \vee (z_{13} \wedge z_{12} \wedge (z_{11} \vee z_{10} \vee \cdots \vee z_1 \vee z_0))$$

앞서 언급한 3가지 경우에 대하여 업데이트 되는

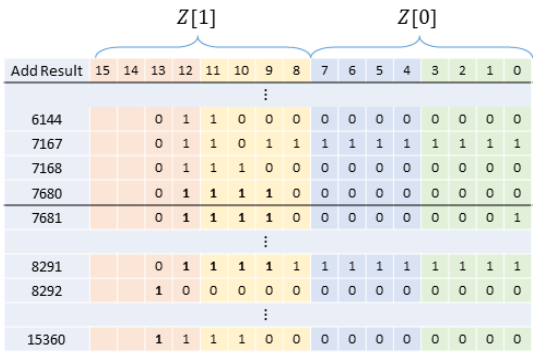


Fig. 1. The bit representation of addition result ($q=7681$)

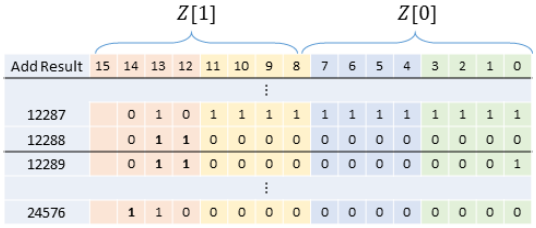


Fig. 2. The bit representation of addition result ($q=12289$)

$Z[0]$는 모듈러 연산이 발생하지 않는 경우 $Z[0]$, 모듈러 연산이 발생하는 경우 $Z[0]-(q\wedge 0xff)$로 나눌 수 있다. 따라서 업데이트 되는 $Z[0]$는 아래와 같이 표현 할 수 있다.

$$Z[0]\leftarrow Z[0]-(q\wedge 0xff)\times Zflag$$

또한 $Z[1]$의 값은 다음과 같이 표현 할 수 있다.

$$(Zflag\wedge(1-Z_0flag))(Z[1]-(q\wedge 0xff00)\times Zflag-1)$$
$$+((1-Zflag)\vee Z_0flag)(Z[1]-(q\wedge 0xff00)\times Zflag)$$

$Zflag$와 Z_0flag에 따라 $(Zflag\wedge(1-Z_0flag))$ 및 $((1-Zflag)\vee Z_0flag)$ 값은 Table 1.과 같이 표현된다. 표의 F는 거짓 (즉, 0을 의미)을 나타하며 T는 진실(즉, 1을 의미)을 나타낸다. Table 1.에서 $(Zflag\wedge(1-Z_0flag))$ 및 $((1-Zflag)\vee Z_0flag)$의 진리 값은 각각 4열과 5열에 $A\wedge D$, $B\vee C$로 나타나 있다. Table 1.에서 볼 수 있듯이 $Zflag=0$의 경우(즉, case 1) $A\wedge D$는 F, $B\vee C$는 T 이므로 $Z[1]$은 처음 $Z[1]$ 값으로 할당 된다. 또한 $Zflag=1, Z_0flag=0$인 경우(즉, case 2) $A\wedge D$는 T, $B\vee C$는 F 이므로 $Z[1]-(q\wedge 0xff00)-1$으로, $Zflag=1, Z_0flag=1$인 경우(즉, case 3) $A\wedge D$는 F, $B\vee C$는 T 이므로 $Z[1]-(q\wedge 0xff00)$으로 업데이트된다.

Table 1. Truth Table associated with $Zflag$ and Z_0flag

$Zflag$ $(=A)$	Z_0flag $(=B)$	$1-Zflag$ $(=C)$	$1-Z_0flag$ $(=D)$	$A\wedge D$	$B\vee C$
F	F	T	T	F	T
F	T	T	F	F	T
T	F	F	T	T	F
T	T	F	F	F	T

Algorithm 4.는 앞쪽의 설명된 성질을 만족하는 CCSPA에 안전한 모듈러 덧셈을 나타내고 있다. Algorithm 4.의 1단계는 $0\le A, B\le q-1$을 만족하는 두 상수 A, B에 대한 덧셈을 시행하는 단계이며 2단계는 Z_0flag 및 $Zflag$를 계산하는 단계이다. 이후 계산된 Z_0flag 및 $Zflag$을 이용하여 3~4단계에서 $Z[0]$ 및 $Z[1]$을 업데이트 한다.

IV. 실험 결과

실험은 두 가지 관점에서 진행되었다. 첫 번째 실험에서 우리는 조건문이 없는 모듈러 덧셈이 동작할 때 실제 모듈러 연산 발생 여부에 따라 차이점이 존재하지 않는지 확인하였다. 다음으로 Lyubashevsky 암호와 Roy 암호의 복호화 부분에 제안하는 CCSPA에 안전한 모듈러 덧셈을 적용한 경우의 성능을 대응법이 적용되지 않은 암호와 비교하였다.

4.1 CCSPA에 안전한 모듈러 덧셈 연산 구별 실험

4.1.1 실험 환경

제안하는 대응법이 적용된 모듈러 덧셈이 모듈러 연산 발생 여부에 따라 차이점이 존재하지 않는지 확인하기 위해 NewAE Technology의 ChipWhisperer - Lite를 이용하여 $q=7681=0x1E01$인 경우에 대하여 실험을 진행하였다.

ChipWhisperer - Lite에서 사용하는 공격 대상 칩은 XMEGA128D4로 8/16-bit AVR XMEGA 마이크로컨트롤러이다. 신호는 7.37MHz 클록 주파수(clock frequency)로 모듈러 덧셈이 실행 될 때 발

Algorithm 4 Secure Modular Addition

Input: $A, B \in [\![0, q-1]\!]$
Output: $Z\,(= Z[1] \parallel Z[0]) \in [\![0, q-1]\!]$

1: $Z \leftarrow \text{Addition}\,(A, B)$
2: Compute $\boldsymbol{Zflag}$ of Z and $\boldsymbol{Z_0flag}$ of Z[0]
$\boldsymbol{Z_0flag} \leftarrow z_7 \vee \cdots \vee z_1 \vee z_0$
$\boldsymbol{Zflag} \leftarrow z_{13} \vee (z_{12} \wedge z_{11} \wedge z_{10} \wedge z_9 \wedge (z_8 \vee \boldsymbol{Z_0flag}))$
3: $Z[1] \leftarrow (\boldsymbol{Zflag} \wedge (1 - \boldsymbol{Z_0flag})) \times (Z[1] - (q \wedge 0xff00) \times \boldsymbol{Zflag} - 1)$
$+ ((1 - \boldsymbol{Zflag}) \vee \boldsymbol{Z_0flag}) \times (Z[1] - (q \wedge 0xff00) \times \boldsymbol{Zflag})$
4: $Z[0] \leftarrow Z[0] - (q \wedge 0xff) \times \boldsymbol{Zflag}$
5: **return** Z

생 되는 전력신호를 29.5MHz 샘플링 레이트로 획득하였다.

4.1.2 실험 결과

Fig. 3.은 동일 입력에 대한 대응법을 적용하지 않은 모듈러 덧셈(Fig. 3. 위) 및 제안하는 CCSPA에 안전한 모듈러 덧셈(Fig. 3. 아래)이 각 16번 동작할 때 수집된 전력으로 가로축은 시간, 세로축은 전력 크기를 나타낸다. 4번째(모듈러 연산 미발생, 0x1A91+0x0202), 10번째(모듈러 연산 발생, 0x1972+0x0899) 덧셈이 동작한 부분을 표시하였으며, 그림에서 나타내고 있는 숫자(Fig. 3. 위의 284 및 296, Fig. 3. 아래의 756)는 하나의 모듈러 덧셈이 시행 될 때 수집한 포인트의 수이다. 또한 그림의 X 및 O는 각각 모듈러 연산 미발생, 모듈러 연산 발생을 의미한다. 그림에서 볼 수 있듯이 기존 모듈러 덧셈에서는 두 부분의 시간차이가 존재하나 제안하는 모듈러 덧셈에서는 동일한 시간에 동작함을 확인 하였다. 즉, 기존 덧셈에서는 모듈러 연산 발생 여부에 따라 포인트의 차이가 발생하나 제안하는 모듈러 덧셈은 모듈러 연산 발생 여부에 상관없이 항상 일정한 값을 지닌다.

Algorithm 5 Decryption of Lyubashevsky cryptosystem

Input: Ciphertext (c_1, c_2), Secret Key r_2
Output: Encoded message m

1: $c_1 \leftarrow \text{Convolution Mul}\,(c_1, r_2)$
2: $\bar{m} \leftarrow \text{Coefficient-wise Add}\,(c_1, c_2)$
3: $m \leftarrow \text{Decode}\,(\bar{m})$
4: **return** m

4.2 성능비교 실험

4.2.1 실험 환경

성능 확인을 위해 ATxmega128A1 프로세서를 사용하는 테스트 보드를 이용하였다. ATxmega128A1 프로세서는 최대 32MHz 주파수 및 128 KB 플래시 메모리와 8 KB SRAM이 포함되어 있다. 성능 확인을 위한 구현은 128비트 보안레벨을 만족하는 경우에 대하여 디코딩을 제외한 복호화 부분을 모두 C 언어를 사용해 구현하였다. 즉, $n = 256$, $q = 7681$을 사용하였다.

실험에 사용된 알고리즘은 Algorithm 1. 및 Algorithm 5.와 같다. CCSPA에 안전한 모듈러 덧셈은 각 알고리즘의 2단계에서 뿐만 아니라 곱 연산 또는 역 NTT 연산에 사용되는 모든 모듈러 덧셈에 적용시켜 비교하였다.

4.2.2 실험 결과

Table 2.는 Lyubashevsky 암호와 Roy 암호의 복호화 연산에 대하여 대응법 적용 여부에 따른 실행 시간을 나타낸 표이다. Lyubashevsky 암호는 대응법이 적용 되지 않았을 때 약 85.51×10^6 cycle이 소모되었으며 대응법을 적용 시켜 복호화를 수행했을 때에는 기존보다 약 11.2% 증가한 95.11×10^6 cycle이 소모되었다.

또한 Roy 암호는 대응법이 적용 되지 않았을 때 약 1.96×10^6 cycle이 소모되었으며 대응법을 적용 시켜 복호화를 수행했을 때에는 기존보다 약 14.9% 증가한 2.25×10^6 cycle이 소모되었다.

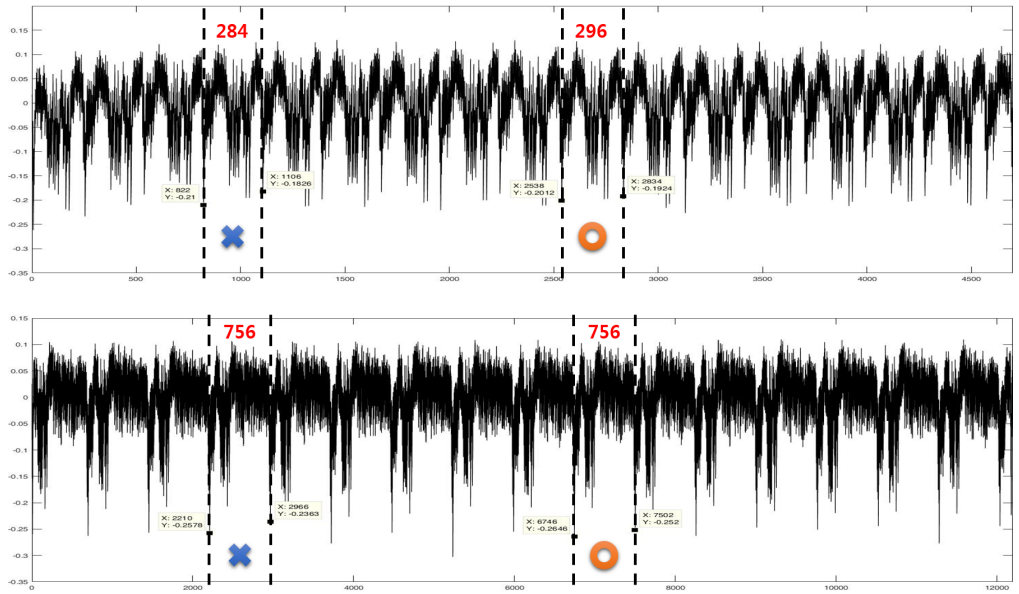


Fig. 3. Top: Power consumption trace of 16 non-secure modular additions. Bottom: Power consumption trace of 16 secure modular additions

Table 2. Performance comparison on ATxmega128

Implementation	Lyubashevsky cryptosystem [8]	
	without countermeasure	with countermeasure
Cycle	85,510,088	95,109,369
Implementation	Roy cryptosystem [9]	
	without countermeasure	with countermeasure
Cycle	1,957,483	2,249,623

V. 결 론

본 논문에서는 격자 기반 암호의 LWE 문제의 대수적 변종인 ring-LWE 문제를 이용한 공개키 암호시스템에 대하여 부채널 분석 연구를 진행하였다. 우선 Roy 암호에만 적용되었던 시간 기반 선택 암호문 SPA 공격이 암호문 c_1의 변형을 통해 Lyubashevsky 암호에 적용 되어 질 수 있음을 보였고, CCSPA 공격에 대응하기 위한 CCSPA에 안전한 모듈러 덧셈을 제안하였다.

현재 ring-LWE 기반 공개키 암호시스템의 부채널 분석 연구는 2015년부터 시작되었다. 특히 Roy 암호에 대한 연구가 주를 이룰 뿐 Lyubashevsky 암호에 대한 연구는 아직 미비하다. 향후 Roy 암호의 고차 부채널 취약점 연구와 함께 Lyubashevsky 암호에 대한 부채널 분석 관점 연구가 진행 되어야 할 것이다.

References

[1] P. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," Proceedings of the 35th Annual Symposium on Foundations of Computer Science, pp. 124-134, Nov. 1994.

[2] P. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," Proceedings of the 16th Annual International Cryptology Conference, pp. 104-113, Aug. 1996.

[3] C. Chen, T. Eisenbarth, I.V. Maurich, and R. Steinwandt, "Differential Power

Analysis of a McEliece Cryptosystem," Proceedings of the 13th International Conference on Applied Cryptography and Network Security, pp. 538-556, Jun. 2015.

[4] M.K. Lee, J.E. Song, D.H. Choi, and D.G. Han, "Countermeasures against the power analysis attack for the NTRU public key cryptosystem," IEICE Transactions on Fundamentals of Electronics on Communications and Computer Sciences, vol.E93-A, no.1, pp.153 - 163, Jan. 2010.

[5] O. Reparaz, S. Roy, F. Vercauteren, and I. Verbauwhede, "A masked ring-LWE implementation," Proceedings of the 17th Workshop on Cryptographic Hardware and Embedded Systems, pp. 683-702, Sep. 2015.

[6] O. Reparaz, R. de Clercq, S. Roy, F. Vercauteren, and I. Verbauwhede, " Additively homomorphic ring-LWE masking," Proceedings of the 7th International Conference on Post-Quantum Cryptography, pp. 233-244, Feb. 2016.

[7] A. Park and D.G. Han, "Chosen ciphertext Simple Power Analysis on software 8-bit implementation of ring-LWE encryption," Proceedings of the Hardware-Oriented Security and Trust (AsianHOST), pp. 1 - 6, Dec. 2016.

[8] V. Lyubashevsky, C. Peikert, and O. Regev, "On Ideal Lattices and Learning with Errors over Rings," Proceedings of the 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 1 - 23, Jun. 2010.

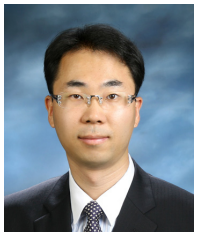
[9] S. Roy, F. Vercauteren, N. Mentens, D. Chen, and I. Verbauwhede, "Compact ring-LWE cryptoprocessor," Proceedings of the 16th Workshop on Cryptographic Hardware and Embedded Systems, pp. 371-391, Sep. 2014.

[10] N. Göttert, T. Feller, M. Schneider, J. Buchmann, and S. Huss, "On the Design of Hardware Building Blocks for Modern Lattice-Based Encryption Schemes," Proceedings of the 14th Workshop on Cryptographic Hardware and Embedded Systems, pp. 512-529, Sep. 2012.

〈저 자 소 개〉

박 애 선 (Aesun Park) 학생회원
2011년 2월: 국민대학교 수학과 학사
2013년 2월: 국민대학교 수학과 석사
2014년 3월~현재: 국민대학교 금융정보보안학과 박사과정
〈관심분야〉 부채널 분석 및 대응법, 스마트 카드 평가, Post-quantum cryptography 등

원 유 승 (Yoo-Seung Won) 학생회원
2012년 2월: 국민대학교 수학과 학사
2014년 2월: 국민대학교 수학과 석사
2014년 3월~현재: 국민대학교 금융정보보안학과 박사과정
〈관심분야〉 정보보호, 부채널 분석, 대칭키 암호 알고리즘, 스마트 카드 보안



한 동 국 (Dong-Guk Han) 종신회원
1999년 2월: 고려대학교 수학과 졸업(학사)
2002년 2월: 고려대학교 수학과 석사 (이학석사)
2005년 2월: 고려대학교 정보보호대학원 박사 (공학박사)
2004년 4월~2005년 4월: 일본 Kyushu Univ., 방문연구원
2005년 4월~2006년 4월: 일본 Future Univ.-Hakodate, Post.Doc.
2006년 6월~2009년 2월: 한국전자통신연구원 정보보호연구단 선임연구원
2009년 3월~현재: 국민대학교 수학과 부교수
〈관심분야〉 공개키 암호시스템 안전성 분석 및 고속 구현, 부채널 분석 및 대응법 설계, IoT 정보보호 기술