Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.32, NO.5, Oct. 2022 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2022.32.5.765

Meta-PKE 구조에 의한 SABER 알고리즘의 임시 키 재사용 공격*

이 창 원,[1†] 전 찬 호,[1] 김 수 리,[2] 홍 석 희[3‡]
[1,3]고려대학교 (대학원생, 교수), [2]성신여자대학교 (교수)

Ephemeral Key Reuse Attack of the SABER Algorithm by Meta-PKE Structure*

Changwon Lee,[1†] Chanho Jeon,[1] Suhri Kim,[2] Seokhie Hong[3‡]
[1,3]Korea University (Graduate student, Professor),
[2]Sungshin Women's University (Professor)

요 약

NIST PQC 표준화 Round 3에 제시된 PKE/KEM 알고리즘인 SABER 알고리즘은 격자 기반 문제 중 Module-LWR 문제를 기반으로 하는 알고리즘으로 Meta-PKE 구조로 되어 있다. 이때, 암호화 과정에서 사용되는 비밀 정보를 임시 키라고 부를 것이며 본 논문에서는 Meta-PKE 구조를 활용한 임시 키 재사용 공격에 대해 설명한다. NIST에서 요구하는 보안 강도 5, 3, 1을 만족하는 각각의 파라미터에 대해 4, 6, 6번의 쿼리를 사용하여 공격한 선행 연구에 대해 자세한 분석과 함께 이를 향상하여 3, 4, 4번의 쿼리만 사용하는 방법을 제시한다. 그리고 추가로 한 번의 쿼리를 통해 임시 키를 복원하는 계산 복잡도를 n차 격자 위에서 각각의 파라미터에 대해 전수조사 복잡도인 $2^{7.91\times n}$,$2^{10.51\times n}$, $2^{12.22\times n}$에서 $2^{4.91\times n}$,$2^{6.5\times n}$,$2^{6.22\times n}$으로 감소시키는 방법을 소개하며 그에 대한 결과 및 한계점을 제시한다.

ABSTRACT

The SABER algorithm, a PKE/KEM algorithm presented in NIST PQC Standardization Round 3, is an algorithm based on the Module-LWR problem among lattice-based problems and has a Meta-PKE structure. At this time, the secret information used in the encryption process is called a ephemeral key, and in this paper, the ephemeral key reuse attack using the Meta-PKE structure is described. For each parameter satisfying the security strengths required by NIST, we present a detailed analysis of the previous studies attacked using 4, 6, and 6 queries, and improve them, using only 3, 4, and 4 queries. In addition, we introduce how to reduce the computational complexity of recovering ephemeral keys with a single query from the brute-force complexity on the n-dimension lattice, $2^{7.91\times n}$,$2^{10.51\times n}$, $2^{12.22\times n}$ to $2^{4.91\times n}$,$2^{6.5\times n}$,$2^{6.22\times n}$, for each parameter, and present the results and limitations.

Keywords: PQC, KEM, Lattice, SABER, Reuse Attack

Received(07. 15. 2022), Modified(09. 05. 2022), Accepted(09. 05. 2022)
* 이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2022R1F1A106361111).

* 본 논문은 2022년도 한국정보보호학회 하계학술대회에 발표한 우수논문을 개선 및 확장한 것임.

† 주저자, lchwon@korea.ac.kr
‡ 교신저자, shhong@korea.ac.kr(Corresponding author)

I. 서 론

1978년, 인수분해 문제의 어려움에 기반한 RSA가 공개되었고 뒤이어 많은 공개키 암호 시스템이 개발되었다. 대부분은 RSA와 마찬가지로 인수분해 문제의 어려움에 기반하거나 이산대수 문제의 어려움에 기반하였으며 그들은 현재 많은 분야에서 사용되고 있다. 그러나 1994년에 Shor가 제안한 양자 알고리즘[1]에 의해 그러한 문제들이 다항 시간 내에 해결될 수 있음이 증명되었다. 이에 따라 양자 컴퓨터 환경에서도 안전한 새로운 공개키 암호 시스템을 개발하기 위해 NIST (미국 국립표준기술 연구소)에서 PQC (Post Quantum Cryptography, 후양자암호)의 표준화 작업을 2017년에 시작하게 되었다.

키 교환. 둘 혹은 그 이상의 집합 간에 통신을 위해 진행하는 중요한 절차인 키 교환 과정은 현재 암호학에서 핵심 분야 중 하나이다. 현재 이산대수 문제를 기반으로 한 DH (Diffie-Hellman) 키 교환 프로토콜 혹은 이를 변형하여 타원곡선을 사용한 ECDH 등이 많이 사용되고 있으며 이를 대체하기 위해 NIST PQC 표준화 작업에 많은 PKE/KEM (Public Key Encryption/Key Encapsulation Mechanism) 알고리즘들이 제시되었다.

현재 NIST PQC 표준화 알고리즘으로 PKE/KEM 알고리즘에서는 round 3에서 제시되었던 4개의 알고리즘 중 CRYSTALS-KYBER[2]가 선정되었으며 새롭게 round 4를 진행하고 있다. SABER[3] 알고리즘은 round 3에서 제시되었던 알고리즘 중 하나로 CRYSTALS-KYBER와 비슷한 구조를 가지고 있으며 Module-LWR 문제를 기반으로 하고 있다.

임시 키. 격자 기반 키 캡슐화 과정에서 키를 생성하는 A와 암호화를 진행하는 B 모두 임의의 난수를 선택하여 각자의 비밀 정보로 삼는다. A는 이러한 비밀 정보를 비밀 키로써 사용하며 B는 암호화 과정에서 난수성을 추가하기 위해 사용하는데 이때 B의 비밀 정보를 임시 키라고 부를 것이다. 네트워크 통신에서 키 또는 난수는 몇몇 재사용 되는 방식이 사용되었다. TLS 1.3[4]에서는 Pre-Shared Key (PSK) 방식을 사용하고 있으며 이는 통신하는 중간에 비밀 키와 공개 키를 재사용하는 것이 허가된 방식이다. 이전부터 이러한 비밀 정보의 재사용에 관련된 연구는 많이 진행되었으며 Liu et al. 은 격자 기반의 NewHope[5]에 대한 임시 키 재사용 공격을 소개하였다[6].

그 이후 [7]에서는 Meta-PKE 구조를 소개하며 많은 격자 기반 암호 알고리즘들이 해당 구조를 따르고 있다고 하였다. 그리고 이러한 Meta-PKE 구조의 취약점으로 임시 키 재사용 공격을 소개하였고 이를 NewHope와 LAC[8]에 적용한 공격을 제시하였다. [9]에서는 이를 바탕으로 하여 SABER와 CRYSTALS-KYBER에 적용한 공격 방법을 소개하였다.

본 논문에서는 격자 기반 PKE/KEM 알고리즘 중 SABER 알고리즘에 대해 이전에 제시된 Meta-PKE 구조를 활용한 임시 키 재사용 공격에 대한 자세한 분석과 함께 향상된 공격 방법에 대해 설명한다.

II. 배경 지식

본 장에서는 SABER 알고리즘의 기본 지식과 Meta-PKE 구조에 대해 설명한다.

Algorithm 1. SABER Algorithm	
Key Generation	
1.	Select $\boldsymbol{A} \leftarrow U(R_q^{k\times k}), \boldsymbol{s} \leftarrow \beta_\mu(R_q^{k\times 1})$
2.	Compute $\boldsymbol{b}=((\boldsymbol{As}+\boldsymbol{h})\bmod q) \gg (\epsilon_q-\epsilon_p)$
3.	Public key : $(\boldsymbol{A},\boldsymbol{b})$, Private key : $\boldsymbol{s}$
Encryption	
1.	Generate a message $m\in R_2$ and select $\boldsymbol{s}' \leftarrow \beta_\mu(R_q^{k\times 1})$
2.	Compute $\boldsymbol{b}'=((\boldsymbol{A}^T\boldsymbol{s}'+\boldsymbol{h})\bmod q) \gg (\epsilon_q-\epsilon_p)$
3.	Compute $v'=\boldsymbol{b}^T(\boldsymbol{s}' \bmod p)$
4.	Compute $c=(v'+h_1-2^{\epsilon_p-1}m \bmod p) \gg (\epsilon_p-\epsilon_T)$
5.	Ciphertext : $(c,\boldsymbol{b}')$
Decryption	
1.	Compute $v=\boldsymbol{b}'^T(\boldsymbol{s} \bmod p)$
2.	Compute $m'=(v-2^{\epsilon_p-\epsilon_T}c+h_2 \bmod p) \gg (\epsilon_p-1)$
3.	Plaintext : m'

Fig. 1. SABER Algorithm

Table 1. Parameters in Saber

	Security Level	n	k	q	p	T	μ
Light Saber	I (AES-128)	256	2	2^{13}	2^{10}	2^3	10
Saber	III (AES-192)	256	3	2^{13}	2^{10}	2^4	8
Fire Saber	V (AES-256)	256	4	2^{13}	2^{10}	2^6	6

2.1 SABER 알고리즘[3]

SABER 알고리즘은 LWE 시스템보다 효율적이라고 알려진 LWR 문제를 변형한 Module-LWR 문제를 기반으로 한 알고리즘이다. 사용되는 집합 $\mathbb{Z}_q$는 정수 q에 대한 modulo ring이고 $\mathbb{Z}_q[x]$는 $\mathbb{Z}_q$위에서의 다항식 ring이며 R_q는 $n=256$인 $\mathbb{Z}_q[x]/(x^n+1)$이다. $\boldsymbol{A}\in R_q^{k\times k}, \boldsymbol{b}\in R_q^{k\times 1}$는 각각 행렬과 벡터를 나타낸다. $x\leftarrow\chi$는 χ 분포에서 x를 샘플링하는 것을 나타낸다. 확률분포는 균등분포인 U와 $P[x \mid x\leftarrow\beta_\mu]=2^{-\mu}\cdot\mu!/(\mu/2+x)!(\mu/2-x)!$의 확률로 $[-\mu/2,\mu/2RIGHT]$에서 x를 샘플링하는 이항분포(CBD) β_μ를 사용한다. $\lfloor\ \rceil$ 연산은 반올림 연산이며 SABER 알고리즘에서는 이를 간단한 shift 연산으로 대체하기 위해 3가지의 상수를 사용한다. 다항식 $h_1\in R_q$는 모든 계수가 $2^{\epsilon_q-\epsilon_p-1}$인 다항식이고, 벡터 $\boldsymbol{h}\in R_q^{k\times 1}$은 각 성분이 h_1과 같은 벡터이며 $h_2\in R_q$는 모든 계수가 $(2^{\epsilon_q-2}-2^{\epsilon_p-\epsilon_T-1}+2^{\epsilon_q-\epsilon_p-1})$인 다항식이다. $\gg$은 bit 단위 shift 연산이다.

SABER 알고리즘은 $\epsilon_q>\epsilon_p>\epsilon_T$인 고정된 상수 $q=2^{\epsilon_q}$, $p=2^{\epsilon_p}$와 고정되지 않은 상수 $T=2^{\epsilon_T}$와 μ를 사용한다. μ는 $\mu<p$로 β_μ에서 비밀 키와 임시 키를 샘플링할 때 사용되는 상수이다. T와 μ는 NIST에서 정한 security level에 따라 값이 다르다(Table 1 참조).

2.2 Meta-PKE

Wang et al.은 몇몇 PKE 알고리즘에서 나타나는 Algorithm 2와 같은 구조를 Meta-PKE라고 명명했다[7]. 이때, Algorithm 2의 암호화 과정 중 5번 과정인 $V=t\times B+f+Y(Y=encode(pt))$의 식이 취약점으로 지적되었다. 이 식에서 V는 암호문, B는 공개키, t,f는 임시 키, Y는 평문이 인코딩된 값이다. [7]에서 공개키가 아래의 정리를 만족할 경우 Y와 t,f값이 Meta-PKE 구조에서 드러날 수 있음을 보여주었다.

정리 1[7]. $t,f,Y\in R_q$이고 각각의 계수들에 대해 $t[i],f[i]\in\{-D,\cdots,D\}$, $Y[i]\in\{0,q/2RIGHT$ $(i=1,\cdots,n)$을 만족하며 $B\in\mathbb{Z}_q$이고 $V=B\times t+f+Y \bmod^{\pm} q$라고 하자. 이때 $2D+1\leqq B<q/4D-1$이면 V값을 통해 t,f,Y의 값을 알 수 있다.

증명. [7] 참조 □

Algorithm 2. Meta-PKE construction	
Key Generation	
1.	Generate a random $seed_A$
2.	Generate $A=Gen(seed_A)$
3.	Sample secrets $sk,d <-\chi$
4.	$B=A\times sk+d$
5.	$pk=(seed_A,B)$
6.	Public key : pk, Private key : sk
Encryption	
1.	Parse $pk=(seed_A,B)$
2.	Generate $A=Gen(seed_A)$
3.	Sample ephemeral keys $t,e,f <-\chi$
4.	$U=t\times A+e$
5.	$V=t\times B+f+encode(pt)$
6.	Return $ct=(U,V)$
Decryption	
1.	Parse $ct=(U,V)$
2.	$W=V-U\times sk$
3.	$pt=decode(W)$
4.	Return pt

Fig. 2. Meta-PKE construction

III. 기존 공격[9] 분석

본 장에서는 [9]에서 제시된 Meta-PKE 구조를 활용한 SABER 알고리즘의 임시 키 재사용 공격에 대해 설명하고 자세히 분석한다. 이때, 임시 키는 암호화 과정의 1번 과정에서 샘플링하는 $\boldsymbol{s}'$을 의미한다. SABER 알고리즘 또한 Meta-PKE 구조로 되어 있으며 각 변수의 형태가 정리 1의 조건에 부합하므로 이를 적용할 수 있다.

정리 2. $\boldsymbol{s}_i, Y(=m\cdot p/2)\in R_q$이고 $\boldsymbol{s}_i$의 계수들이 $\boldsymbol{s}_i[j]\in\{-\mu/2,\cdots,\mu/2\},\mu/2<p$이고 평문에 해당하는 Y의 계수가 $Y[j]\in\{0,p/2RIGHT,i=1,\cdots,n$ 이다. 공개키의 각 원소에 대해 상수항을 제외한 나머지 항들의 계수가 0인 $\boldsymbol{b}_i\in\mathbb{Z}_p$의 형태를 가지고 있으며 $V_i=\boldsymbol{b}_i\times\boldsymbol{s}_i+h_1-Y \bmod p$라고 하자.
위의 값들이 다음의 3가지 조건

① $\boldsymbol{b}_i \gg (\epsilon_p - \epsilon_T) = 1$
② $(p/2 - \mu\boldsymbol{b}_i) \gg (\epsilon_p - \epsilon_T) \geqq 1$
③ $2\mu + 2 \leqq p$
을 만족한다면 $c_i = V_i \gg (\epsilon_p - \epsilon_T)$로부터 $\boldsymbol{s}_i$와 Y값이 드러난다.

증명. $V_i = \boldsymbol{b}_i \times \boldsymbol{s}_i + h_1 - Y \bmod p$에서 h_1과 $\boldsymbol{b}_i$는 각각 상수와 공개키이다. 만약 $c_i = V_i \gg (\epsilon_p - \epsilon_T)$의 값이 $\boldsymbol{s}_i, Y$에 대해 일대일 함수의 관계를 가질 경우 $\boldsymbol{s}_i, Y$의 값을 c_i의 값을 통해 구별할 수 있다. 위의 조건들을 만족할 때 c_i가 $\boldsymbol{s}_i, Y$에 대해 일대일 함수 관계를 만족한다는 것을 증명하기 위해 $\boldsymbol{s}_i$, $\boldsymbol{s}'_i$, Y, Y'에 대한 두 식 $V_i = \boldsymbol{b}_i \times \boldsymbol{s}_i + h_1 - Y \bmod p$와 $V_i' = \boldsymbol{b}_i \times \boldsymbol{s}'_i + h_1 - Y' \bmod p$를 생각하자. $\boldsymbol{s}_i[j]$와 $\boldsymbol{s}'_i[j]$이 서로 다르다면 $V_i[j]$와 $V_i'[j]$의 차이의 최솟값은 $\boldsymbol{b}_i$이다. 그러므로 $\boldsymbol{b}_i \gg (\epsilon_p - \epsilon_T) = 1$을 만족할 경우, 암호문 $c_i[j] \neq c'_i[j]$이고 $c_i[j] - c'_i[j] = 1$이 성립한다. 만약 Y와 Y'의 값 또한 다르다면 V_i와 V_i'의 차이는 $(p/2 - \mu\boldsymbol{b}_i)$의 최솟값을 가지므로 이에 따라 $(p/2 - \mu\boldsymbol{b}_i) \gg (\epsilon_p - \epsilon_T) \geqq 1$일 때 $c_i[j] \neq c'_i[j]$가 된다. 그리고 V_i의 상의 크기가 $\mathbb{Z}_p$의 크기보다 작아야 하므로 $2\mu + 2 \leqq p$이어야 한다. □

원래의 공개키는 R_q에 속해 있는 다항식을 원소로 하는 k차 벡터이지만 위의 정리를 사용하기 위해 아래의 내용에서는 상수항을 제외한 나머지 계수가 0인 형태를 가진 공개키만을 사용한다. 즉 $\boldsymbol{b}_i$들이 다항식이 아닌 상수항만 남아있는 $\boldsymbol{b}_i \in \mathbb{Z}_q$의 형태를 가진 공개키만 사용하였고 이를 통해 정리 2를 사용하여 [9]에서 임시 키 재사용 공격을 SABER에 적용하였다. NIST에서 요구하는 보안 강도에 따른 각 파라미터인 FireSaber, Saber, LightSaber에 대해, 각 쿼리마다 암호화를 진행할 때 같은 임시 키를 재사용하는 상황에서 각각 4, 6, 6번의 쿼리를 통해 임시 키를 복원하였다. FireSaber의 경우 위의 정리를 만족하는 $\boldsymbol{b}_i$를 쉽게 찾을 수 있어 공격이 단순하지만, 나머지의 경우 그렇지 않아 추가적인 방법을 더 사용하여야 했다. 이에 대한 분석을 위해 다음과 같은 따름정리를 사용한다.

따름정리 1. 정리 2의 ① 조건을 $\boldsymbol{b}_i \gg (\epsilon_p - \epsilon_T) \geqq 1$로 변경해도 $c_i = V_i \gg (\epsilon_p - \epsilon_T)$로부터 $\boldsymbol{s}_i$와 Y값을 드러난다. 그리고 $\boldsymbol{b}_i \gg (\epsilon_p - \epsilon_T) = 1$일 때 같은 Y값에 대해 $\boldsymbol{s}_i$와 $\boldsymbol{s}_i + 1$로 계산된 암호문 c_i 값의 차이는 정확히 1이며 $r < \mu$인 정수 r에 대해 $r\boldsymbol{b}_i \gg (\epsilon_p - \epsilon_T) = 0$이면 평문이 같을 때 $\boldsymbol{s}_i$와 $\boldsymbol{s}_i + r$로 계산된 암호문 c_i값은 같다.

따름정리 2. $\boldsymbol{s}_i, Y\left(= \frac{p}{2}m\right) \in R_q$이고 $\boldsymbol{s}_i$의 계수들이 $\boldsymbol{s}_i[j] \in \{-\mu/2, \cdots, \mu/2\}, \mu/2 < p$이고 평문에 해당하는 Y의 계수가 $Y[j] \in \{0, p/2 RIGHT; i = 1, \cdots, n\}$ 이다. 공개키의 각 원소에 대해 상수항을 제외한 나머지 항들의 계수가 0인 $\boldsymbol{b}_i \in \mathbb{Z}_p$의 형태를 가지고 있으며 $V_i = \boldsymbol{b}_i \times \boldsymbol{s}_i + h_1 - Y \bmod p$라고 하자. 그리고 암호문은 $c_i = V_i \gg (\epsilon_p - \epsilon_T)$과 같이 계산된다.
이때, $r < \mu$인 상수 r에 대해 조건 $(p/2 - r\boldsymbol{b}_i) \gg (\epsilon_p - \epsilon_T) \geqq 1$을 만족하면 평문이 다를 때 $r_1 \leqq |r|$인 r_1에 대해 $\boldsymbol{s}_i$에 의해 계산된 암호문 c_i의 값은 $\boldsymbol{s}_i + r_1$에 의해 계산된 암호문 c_i의 값과 다르다. 그리고 $(p/2 - r\boldsymbol{b}_i) \gg (\epsilon_p - \epsilon_T) = 0$이면 평문이 다를 때 $\boldsymbol{s}_i$에 의해 계산된 암호문 c_i의 값은 $\boldsymbol{s}_i + r$에 의해 계산된 암호문 c_i의 값과 같다.

공개키들은 모두 상수항만 남은 다항식을 사용하기 때문에 다항식 $\boldsymbol{s}'_i$의 각 계수는 차수가 같은 항을 제외한 다른 항에는 영향을 미치지 않게 된다. 그리고 공개키 원소 중 $\boldsymbol{b}_i$만 정리 2를 만족하는 값을 가지고 $\boldsymbol{b}_j (j \neq i)$는 0이면 암호문에는 $\boldsymbol{s}'_i$만 영향을 미치기 때문에 정리 2를 통해 구별할 수 있게 된다. 이를 사용하여 FireSaber에서는 [16,0,0,0], [0,16,0,0], [0,0,16,0], [0,0,0,16] 의 $\boldsymbol{b}_i = 16$인 공개키를 사용하여 $\boldsymbol{s}'_i$를 복원할 수 있다(Algorithm 4 참조).

그러나 Saber와 LightSaber에서는 정리 2를 만

Algorithm 3. SABER Enc Oracle ($O_{Enc}(\boldsymbol{A}, \boldsymbol{b}, m)$)
Input : $\boldsymbol{A}$, $\boldsymbol{b}$, m
1. $v = \boldsymbol{b}^T(\boldsymbol{s}' \bmod p)$
2. $c = (v + h_1 - 2^{\epsilon_p - 1} m \bmod p) \gg (\epsilon_p - \epsilon_T)$
3. return c

Fig. 3. SABER Encryption Oracle(Using fixed ephemeral key)

Algorithm 4. FireSaber key reuse attack

Input : $\boldsymbol{A}$, m

$B = 2^4$
for i from 0 to 4 do
  $\boldsymbol{b}_i = []$
  for j from 0 to 4 do
    if $j == i$ then
      $\boldsymbol{b}_i$.append(B)
    else
      $\boldsymbol{b}_i$.append(0)
  $c = O_{Enc}(\boldsymbol{A}, \boldsymbol{b}_i, m)$
  for l from 0 to n do
    if $T/2 - \mu \leq c[l] \leq T/2 + \mu$ then
      $\boldsymbol{s}'_i[l] = c[l] - T/2$
    else if $c[l] \leq \mu$ then
      $\boldsymbol{s}'_i[l] = c[l]$
    else
      $\boldsymbol{s}'_i[l] = c[l] - T$
return $\boldsymbol{s}'$

Fig. 4. FireSaber key reuse attack algorithm

족하는 $\boldsymbol{b}_i$를 찾을 수 없어 추가적인 계산을 통해 공격을 진행하였다.

3.1 Saber

Saber에서는 따름정리 1을 사용하면 $\boldsymbol{b}_i \gg 6 \geq 1$이어야 하고 이는 $\boldsymbol{b}_i \geq 64$가 된다. 그리고 ②번 조건에서 $\left(2^{10}/2 - 2 \times 4\boldsymbol{b}_i\right) \gg 6 \geq 1$이고 $\boldsymbol{b}_i \leq 56$이어서 조건을 만족하는 $\boldsymbol{b}_i$를 찾을 수 없다. 그러나 $\boldsymbol{b}_i = 64$는 $\left(2^{10}/2 - 7\boldsymbol{b}_i\right) \gg 6 \geq 1$를 만족하며 이는 따름정리 2에 따라 $\boldsymbol{s}'_i$의 차이가 7 이하일 때 평문이 다르면 암호문의 값이 모두 다른 값을 가짐을 의미한다. 즉 Saber에서는 $\boldsymbol{s}'_i$가 -4와 4일 때를 제외하고는 다른 값을 가지게 된다. 그리고 따름정리 1의 조건을 만족하고 있으므로 $\boldsymbol{s}'_i$의 값이 각각 -4와 4인 경우를 제외하고는 모두 암호문을 보고 구별할 수 있게 된다.

이제 $\boldsymbol{s}'_i$값이 -4 또는 4일 때를 구별하기 위해 64에서 32를 더한 96을 $\boldsymbol{b}_i$로 사용하게 되면 4일 때는 $\boldsymbol{b}_i = 64$에서 V_i의 값에 $4 \times 32 = 2^7$만큼의 값이 더해지며 -4일 때는 2^7만큼의 값이 빼지게 된다. 즉 둘의 차이는 2^8이 되며 이는 2^{10}보다 작아 $\bmod p$를 거쳐도 다른 값이 되며 또한 2^9보다도 작아 평문 값이 다를 때 V_i값이 다르며 또한 2^6보다는 크기 때문에

Algorithm 5. Saber key reuse attack

Input : $\boldsymbol{A}$, m

$B = 2^6$
for i from 0 to 3 do
  $\boldsymbol{b}_i = []$
  for j from 0 to 3 do
    if $j == i$ then
      $\boldsymbol{b}_i$.append(B)
    else
      $\boldsymbol{b}_i$.append(0)
  $c = O_{Enc}(\boldsymbol{A}, \boldsymbol{b}_i, m)$
  for l from 0 to n do
    if $c[l] == 4$ or $c[l] == 12$ then
      continue
    else if $5 \leq c[l] \leq 11$ then
      $\boldsymbol{s}'_i[l] = c[l] - T/2$
    else if $0 \leq c[l] \leq 3$ then
      $\boldsymbol{s}'_i[l] = c[l]$
    else
      $\boldsymbol{s}'_i[l] = c[l] - T$
$B = 2^6 + 2^5$
for i from 0 to 3 do
  $\boldsymbol{b}_i = []$
  for j from 0 to 3 do
    if $j == i$ then
      $\boldsymbol{b}_i$.append(B)
    else
      $\boldsymbol{b}_i$.append(0)
  $c = O_{Enc}(\boldsymbol{A}, \boldsymbol{b}_i, m)$
  for l from 0 to n do
    if $c[l] == 2$ or $c[l] == 10$ then
      $\boldsymbol{s}'_i[l] = -4$
    else if $c[l] == 6$ or $c[l] == 14$ then
      $\boldsymbol{s}'_i[l] = 4$
return $\boldsymbol{s}'$

Fig. 5. Saber key reuse attack algorithm

shift 연산 이후에도 그 차이가 유지가 된다. 즉 $\boldsymbol{s}'_i$값이 -4와 4일 때 c_i값이 달라지고 이를 통해 $\boldsymbol{s}'_i$값을 구별할 수 있다. 이를 이용하여 Saber에서는 rank 값이 3이므로 $\boldsymbol{b}_i = 64$와 $\boldsymbol{b}_i = 96$을 각 i에 대해 적용하여 6번의 쿼리를 통해 $\boldsymbol{s}'_i$를 복원할 수 있다(Algorithm 5 참조).

3.2 LightSaber

LightSaber의 경우 마찬가지로 따름정리 1로

Table 2. Value of $c[j]$ according to $\boldsymbol{s'}_1[j]$ and $m[j]$ in FireSaber when $\boldsymbol{b}=[2^6,2^3,0,0]$

$\boldsymbol{s'}_1[j]$	-3	-2	-1	0	1	2	3
$m[j]=0$	50, 54, 58, 62, 2, 6, 10	51, 55, 59, 63, 3, 7, 11	51, 55, 59, 63, 3, 7, 11	52, 26, 60, 0, 4, 8, 12	52, 26, 60, 0, 4, 8, 12	53, 57, 61, 1, 5, 9, 13	53, 57, 61, 1, 5, 9, 13
$m[j]=1$	18, 22, 26, 30, 34, 38, 42	19, 23, 27, 31, 35, 39, 43	19, 23, 27, 31, 35, 39, 43	20, 24, 28, 32, 36, 40, 44	20, 24, 28, 32, 36, 40, 44	21, 25, 29, 33, 37, 41, 45	21, 25, 29, 33, 37, 41, 45

$\boldsymbol{b}_i \gg 7 \geq 1$이어야 하고 이는 $\boldsymbol{b}_i \geq 128$가 되며 정리 2의 ②번 조건에서 $\left(2^{10}/2-2\times 5\boldsymbol{b}_i\right)\gg 7 \geq 1$이며 이는 $\boldsymbol{b}_i \leqq 38$이어서 조건을 만족하는 $\boldsymbol{b}_i$를 찾을 수 없다. 이때 $\boldsymbol{b}_i=128$은 $\left(2^{10}/2-3\boldsymbol{b}_i\right)\gg 7 \geq 1$을 만족하고 따름정리 2에 따라 $\boldsymbol{s}_i'$의 차이가 3 이하일 때 암호문의 값은 모두 다른 값을 가지게 된다. 그리고 $\boldsymbol{s'}_i$의 차이가 4일 경우 V_i의 값의 차이는 $\left(2^{10}/2-4\times 128\right)\gg 6=0$이 되어 따름정리 2에 의해 평문이 다를 때 암호문이 같은 값을 가지게 된다. 즉 $\boldsymbol{b}_i=128$을 사용할 경우 $\boldsymbol{s'}_i$, m, c_i에 대한 표에서 연속된 4개의 $\boldsymbol{s}_i$값에 대해서는 평문 값과 상관없이 모두 다른 값이 나오며 차이가 4인 $\boldsymbol{s}_i$값에 대해서는 평문이 다를 때 암호문이 같게 된다. 이제 이를 구분하기 위해 $\boldsymbol{b}_i=16$을 사용한다. 16을 $\boldsymbol{s'}_i$에 곱한 값에서 7만큼 shift를 하게 되면 -5부터 5의 값들의 경우 부호만 구분할 수 있게 된다. 이를 통해 먼저 $\boldsymbol{s'}_i$가 음수인지 양수인지 판단하게 된다. $\boldsymbol{s'}_i$가 음수일 경우 구분이 불가능한 $\boldsymbol{s'}_i$값은 -5와 -1이 있고, 양수일 경우 0과 4, 1과 5가 존재하며 이 값들에 대한 암호문은 평문이 다를 때 각각 같아 이들만 구분할 수 있으면 된다. 이를 해결하기 위해 128에 64를 더한 192를 $\boldsymbol{b}_i$로 사용하게 되면 $\boldsymbol{s'}_i$가 4 차이가 날 때 Saber에서 96을 사용했던 것처럼 V_i의 값의 차이가 2^8이 되게 된다. 이는 마찬가지로 2^{10}과 2^9보다 작은 값이고 2^7보단 큰 값이므로 mod$p$와 shift 연산 이후에도 그 값이 다르게 된다. 이는 $\boldsymbol{b}_i=128$과 $\boldsymbol{b}_i=16$를 이용하여 $\boldsymbol{s'}_i$의 값의 후보를 좁히고 $\boldsymbol{b}_i=192$를 통해 온전하게 구별해내는 과정으로 $\boldsymbol{s'}_i$값을 복원한다. 즉 $\boldsymbol{b}_i=128$, $\boldsymbol{b}_i=16$, $\boldsymbol{b}_i=192$를 사용하여 rank가 2인 LightSaber에서는 6번의 쿼리를 통해 $\boldsymbol{s'}_i$를 복원할 수 있다(Algorithm 6 참조).

IV. 향상된 공격 기법

4.1 향상된 재사용 공격

4.1.1 FireSaber

FireSaber에서 암호문은 mod$p$ 연산을 통해 계산된 2^{10} 이하의 10비트 값에 $\epsilon_p-\epsilon_T=4$만큼 shift 연산을 하여 6비트 값을 만들며 이는 $2^6=64$의 경우의 수를 갖게 된다. 반면 한 번의 암호화에서 $\boldsymbol{s'}_i$는 -3에서 3까지의 값을, m은 0과 1의 값을 가질 수 있으며 이에 대한 경우의 수는 $7\times 2=14$로 하나의 특별한 공개키를 통해 하나의 $\boldsymbol{s'}_i$만을 복원하는 것은 효율적이지 않다. 이런 생각을 확장하여 t번의 쿼리를 통해 암호화를 진행한다고 하자. 그러면 암호문이 가능한 경우의 수는 $2^{6\times t}$인데 FireSaber의 rank는 4이며 $\boldsymbol{s'}_i$ 값은 변하지 않으므로 $\boldsymbol{s'}_i$, m가 가질 수 있는 경우의 수는 $7^4\times 2^t$이 된다. 일대일 대응이 되려면 암호문의 경우의 수가 더 커야 하므로 $7^4\times 2^t < 2^{6\times t}$가 되어야 하고 $t=3$이 이를 만족한다. 이를 통해 FireSaber에서 공격이 가능한 최소 쿼리 수는 3으로 추측할 수 있다. 이전에 FireSaber에 대해 제시된 결과는 4번의 쿼리를 사용하였으며 $\boldsymbol{s'}$의 원소 4개에 대해 각 쿼리마다 한 개의 $\boldsymbol{s'}_i$만을 드러냈었다. 이를 더 적은 3번의 쿼리를 통해 $\boldsymbol{s'}$를 복원하려면 한 번의 쿼리에서 한 개의 $\boldsymbol{s'}_i$ 뿐만 아니라 다른 원소의 정보도 어느 정도 얻을 수 있어야 한다고 추측할 수 있다.

$\boldsymbol{b}$가 $[\boldsymbol{b_0},0,0,0\mathit{RIGHT}]$의 형태라고 하자. 그러면 v'의 계수들의 경우의 수는 $\boldsymbol{s'}_0$에만 영향을 받으므로 7이고 Y까지 고려하여 나타날 수 있는 c의 경우의 수는 14가 된다. 이는 c의 4비트만 봐도 $\boldsymbol{s'}_0$와 Y를 구별할 수 있음을 의미한다. 이때 c는 2^6까지 총 6비트 값으로 출력되므로 2비트의 정보가 필요하지 않음을 알 수 있다. 이때, -3부터 3까지의 값을 mod$p$에 맞게 변경한 뒤 2진수로 변경하였을 때 각 값의

Algorithm 6. LightSaber key reuse attack

Input : $\boldsymbol{A}$, m

```
B = 2^4
nlist = []
for i from 0 to 2 do
    b_i = []
    for j from 0 to 2 do
        if j == i then
            b_i.append(B)
        else
            b_i.append(0)
    c = O_Enc(A, b_i, m)
    for l from 0 to n do
        if c[l] == 7 or c[l] == 3 then
            nlist.append(true)
        else
            nlist.append(false)
B = 2^7
for i from 0 to 2 do
    b_i = []
    for j from 0 to 2 do
        if j == i then
            b_i.append(B)
        else
            b_i.append(0)
    c = O_Enc(A, b_i, m)
    for l from 0 to n do
        if nlist[l] then
            if 4 ≤ c[l] ≤ 6 then
                s'_i[l] = c[l] - 8
            else if 0 ≤ c[l] ≤ 2 then
                s'_i[l] = c[l] - 4
            else
                continue
        else
            if 2 ≤ c[l] ≤ 3 then
                s'_i[l] = c[l]
            else if 6 ≤ c[l] ≤ 7 then
                s'_i[l] = c[l] - 4
            else
                continue
B = 2^7 + 2^6
for i from 0 to 2 do
    b_i = []
    for j from 0 to 2 do
        if j == i then
            b_i.append(B)
        else
            b_i.append(0)
    c = O_Enc(A, b_i, m)
    for l from 0 to n do
        if nlist[l] then
            if c[l] == 0 or c[l] == 4 then
                s'_i[l] = -5
            else if c[l] == 2 or c[l] == 6 then
                s'_i[l] = -1
            else
                continue
        else
            if c[l] == 0 or c[l] == 4 then
                s'_i[l] = 0
            else if c[l] == 1 or c[l] == 5 then
                s'_i[l] = 1
            else if c[l] == 2 or c[l] == 6 then
                s'_i[l] = 4
            else if c[l] == 3 or c[l] == 7 then
                s'_i[l] = 5
            else
                continue
return s'
```

Fig. 6. LightSaber key reuse attack algorithm

하위 3비트는 101_2부터 011_2까지의 값으로 모두 다른 값이 나온다. 즉 암호문의 일부인 3비트에 -3부터 3까지의 값의 하위 3비트를 넣고 다른 영향이 없다면 이를 온전히 구분할 수 있다. 이를 이용하기 위해 $\boldsymbol{b}_0$를 c의 6비트 중 상위 4비트에만 영향을 주는 값으로 선택했다고 하자. 예를 들어 $2^6 = 64$일 때 어떤 값이 곱해져도 후에 mod$p$ 연산으로 최상위 비트부터 11번째 비트까지 아무런 영향 없이 사라지게 되어 상위 4비트에만 영향을 주게 된다. 그러면 Y가 0 또는 2^9이므로 암호문의 최상위 비트에만 영향을 주

Table 3. Value of c according to $\boldsymbol{s'}_0[j]$ and $m[j]$ in FireSaber when $\boldsymbol{b}=[2^6,2^3,0,0]$

$\boldsymbol{s'}_0[j]$	-3	-2	-1	0	1	2	3
$m[j]=0$	50,51,52,53	54,55,56,57	58,59,60,61	62,63,0,1	2,3,4,5	6,7,8,9	10,11,12,13
$m[j]=1$	18,19,20,21	22,23,24,25	26,27,28,29	30,31,32,33	34,35,36,37	38,39,40,41	42,43,44,45

며 이는 $\boldsymbol{s'}_0$를 구분할 때 필요한 $\boldsymbol{s'}_0$의 정보가 온전히 담긴 암호문의 3비트에는 영향을 주지 않게 된다. 이제 $\boldsymbol{s'}_1$이 c에 영향을 주면서도 $\boldsymbol{s'}_0$를 드러내는 3비트에는 큰 영향이 없도록 하는 방법을 생각해보자. -3에서 3을 2진수로 변경한 값들의 하위 2번째, 3번째 비트를 생각해보자. 각각 10_2, 11_2, 00_2, 01_2의 값 중 하나를 갖게 된다. 이때, 10_2, 11_2의 경우 이 위의 상위비트들이 모두 1이고, 00_2, 01_2은 0임을 알 수 있다. 즉, $\boldsymbol{s'}_1$의 상위비트 값이 c에 영향을 준다고 해도 하위 2번째, 3번째 비트 값들이 온전히 드러나 있다면 어떠한 영향을 주었는지 구별할 수 있게 된다. 이전에 $\boldsymbol{b}_0$를 통해 c의 상위 4비트에 $\boldsymbol{s'}_0$를 구별할 수 있는 정보를 담을 수 있었고 이때 하위 2비트는 아무런 값이 들어가지 않는 상태였다. 이제 $\boldsymbol{s'}_1$의 하위 2번째, 3번째 비트가 c의 하위 2비트에 들어가게 하는 $\boldsymbol{b}_1$을 찾아 새롭게 $\boldsymbol{b'}=[\boldsymbol{b}_0,\boldsymbol{b}_1,0,0]$을 구성한다면 c의 하위 2비트를 통해 $\boldsymbol{s'}_1$의 값의 정보를 얻게 된 후 그의 상위비트가 0 또는 1로 되어 있다는 정보를 얻을 수 있다. 이를 통해 $\boldsymbol{s'}_0$의 영향을 받은 상위 4비트의 값을 온전히 분리해낼 수 있고 이를 이용하여 $\boldsymbol{s'}_0$를 온전히 얻을 수 있다. 실제로 $\boldsymbol{b}_0=2^6$, $\boldsymbol{b}_1=2^3$인 공개키 $[2^6,2^3,0,0]$에 대해 c값을 계산한 뒤 Taber 2에서와같이 $\boldsymbol{s'}_1$의 일부 정보를 복원할 수 있다. 또한 $\boldsymbol{s'}_0$의 값에 대한 정보를 얻어 온전히 복원할 수 있게 된다(Table 3 참조).

Algorithm 7. New FireSaber key reuse attack

Input : $\boldsymbol{A}$, m

$r_1=0$
$B_1=2^6$, $B_2=2^3$, $B_3=2^4$, $B_4=2^5$
for i from 0 to 2 do
  $\boldsymbol{b_i}=[0,0,0,0]$
  if $i==0$ then
    $\boldsymbol{b_i}=[B_1,B_2,0,0]$
  else if $i==1$ then
    $\boldsymbol{b_i}=[0,0,B_1,B_2]$
  else
    $\boldsymbol{b_i}=[0,B_3,0,B_4]$
  $c_i=O_{Enc}(\boldsymbol{A},\boldsymbol{b_i},m)$
for j from 0 to $n-1$ do
  for l from 0 to 2 do
    if $c_i[j]\geqq 50$ or $c_i[j]\leqq 13$ then
      $c_i[j]=(c_i[j]+32)\bmod 64$
  $\boldsymbol{s'}_0[j]=((c_0[j]+2)\gg 2)-8$
  $\boldsymbol{s'}_2[j]=((c_1[j]+2)\gg 2)-8$
  if $(c_0[j]\gg 1)\&1==1$
    $\boldsymbol{s'}_1[j]=1\|(c_0[j]\&3)\|(c_2[j]\&1)-16$
  else
    $\boldsymbol{s'}_1[j]=(c_0[j]\&3)\|(c_2[j]\&1)$
  $r_1=c_2[j]-s_1[j]$
  if $(c_1[j]\gg 1)\&1==1$
    $\boldsymbol{s'}_3[j]=1\|((c_1[j]\&3))\|((r_1\gg 1)\&1)-16$
  else
    $\boldsymbol{s'}_3[j]=((c_1[j]\&3))\|((r_1\gg 1)\&1)$
return $\boldsymbol{s'}$

Fig. 7. New FireSaber key reuse attack algorithm

이와 비슷한 과정을 $\boldsymbol{s'}_2$와 $\boldsymbol{s'}_3$에 적용하기 위해 다음 공개키로 $[0,0,2^6,2^3]$을 사용하면 $\boldsymbol{s'}_2$의 온전한 값을 복원할 수 있으며 $\boldsymbol{s'}_1$과 마찬가지로 $\boldsymbol{s'}_3$에 대한 정보도 일부 드러난다. 그리고 이때 $\boldsymbol{s'}_1$, $\boldsymbol{s'}_3$의 경우의 수는 각각 2개씩 총 4가지로 6비트 c값을 통해 충분히 알아낼 수 있을 것으로 추측할 수 있다. 이제 이를 위한 공개키를 찾아보자. 현재 알고 있는 정보는 각각의 하위 2번째, 3번째 비트이고 만약 최하위비트를 알게 될 경우 $\boldsymbol{s'}_1$, $\boldsymbol{s'}_3$의 값을 알 수 있어 각각의 최하위비트가 c의 1비트씩에만 영향을 줄 수 있게 하는 것만으로도 충분하다. $\boldsymbol{b}_1$를 2^4이라고 하면 $\boldsymbol{s'}_1$의 최하위비트는 c의 최하위비트에 영향을 주게 된다. 그리고 $\boldsymbol{b}_3$를 2^5이라고 하면 $\boldsymbol{s'}_3$는 c의 최하위비트에

는 영향을 주지 않게 되므로 공개키를 $[0,2^4,0,2^5]$으로 구성하게 되면 c값을 통해 $\boldsymbol{s'}_1$의 최하위비트를 알 수 있게 되고 이에 따라 이전에 $[2^6,2^3,0,0]$를 통해 얻은 정보와 결합하여 $\boldsymbol{s'}_1$을 복원할 수 있다. 그리고 $\boldsymbol{s'}_1$을 복원함에 따라 $[0,2^4,0,2^5]$을 통해 얻은 c값에서 $\boldsymbol{s'}_1$의 영향력을 지워 $\boldsymbol{s'}_3$의 값 또한 복원할 수 있게 된다(Table 4 참조). 즉 공개키 $[2^6,2^3,0,0]$, $[0,0,2^6,2^3]$, $[0,2^4,0,2^5 RIGHT]$의 공개키를 통한 3번의 쿼리를 진행하여 임시 키 $\boldsymbol{s'}$를 복원할 수 있게 된다.

Table 4. Value of c according to $\boldsymbol{s'}_1[j]$, $\boldsymbol{s'}_3[j]$ $=-2$or -1 and $m[j]$ in FireSaber when $\boldsymbol{b}=[0,2^4,0,2^5]$

$\boldsymbol{s_1'}[j]$	-2	-2	-1	-1
$\boldsymbol{s_3'}[j]$	-2	-1	-2	-1
$m[j]=0$	58	60	59	61
$m[j]=1$	26	28	27	29

4.1.2 Saber & LightSaber

Saber와 LightSaber에서도 FireSaber에 대한 공격과 유사한 과정을 진행하기 위해 먼저 각각의 파라미터에 대해 암호화에서 5번 과정의 식 $c=(v'+h_1-2^{\epsilon_p-1}m \bmod p) \gg (\epsilon_p-\epsilon_T)$에 대한 경우의 수를 계산해 보면 각각 최소 4번의 쿼리만으로도 공격할 수 있음을 추측할 수 있다. Saber와 LightSaber에서는 하나의 공개키로 임시 키의 한 원소를 바로 알아내는 것이 아니라 FireSaber에서 $\boldsymbol{s'}_1$을 구하는 것과 비슷하게 각 원소의 일부 정보를 알아내어 경우의 수를 작게 만든 뒤 한 번 더 연산하여 온전히 복원해내는 과정을 거친다.

Saber에서 $\boldsymbol{b}_i=32$를 사용하게 되면 따름정리 1에서 $\boldsymbol{b}_i \gg (\epsilon_p-\epsilon_T)=32 \gg 6=0$의 식을 만족하며 $2\boldsymbol{b}_i \gg (\epsilon_p-\epsilon_T)=64 \gg 6=1$이고 정리 2의 ②조건을 만족하므로 $\boldsymbol{s'}_i$의 값의 차이가 2 이상일 때 다른 암호문 값을 가진다(Table 5 참조). 그리고 이때 얻을 수 있는 정보가 $\boldsymbol{s'}_0$의 하위 2번째, 3번째 비트가 된다. 이를 각 $\boldsymbol{s'}_i$에 적용하게 되면 3번의 쿼리를 통해 $\boldsymbol{s}_i$의 하위 2번째, 3번째 비트를 알게 되므로 최하위 비트만 알면 $\boldsymbol{s'}$를 복원할 수 있다. 이를 알기 위해 FireSaber와 비슷한 방법을 사용한다. 그래서 $\boldsymbol{s'}_0$는

Table 5. Value of c according to $\boldsymbol{s'}_0[j]$ and $m[j]$ in Saber when $\boldsymbol{b}=[2^5,0,0]$

$\boldsymbol{s'}_0[j]$	-4	-3	-2	-1	0	1	2	3	4
$m[j]=0$	14	14	15	15	0	0	1	1	2
$m[j]=1$	6	6	7	7	8	8	9	9	10

하위 4비트가 순서대로 암호문의 4비트에 영향을 주도록, $\boldsymbol{s'}_1$은 하위 3비트가 암호문의 2번째 비트부터, $\boldsymbol{s'}_2$은 하위 2비트가 암호문의 3번째 비트부터 영향을 주도록 하는 $\boldsymbol{b}_i$들을 사용한다면 암호문을 보고 모두 구별할 수 있게 된다. 이를 만족하는 $\boldsymbol{b}$는 $[2^6,2^7,2^8]$

Algorithm 8. New Saber key reuse attack

Input : $\boldsymbol{A}$, m

$r_1=0,\ r_2=0$
$B_1=2^5,\ B_2=2^6,\ B_3=2^7,\ B_4=2^8$
for i from 0 to 3 do
  $\boldsymbol{b_i}=[0,0,0]$
  if $i==3$ then
    $\boldsymbol{b_i}=[B_2,B_3,B_4]$
  else
    for j from 0 to 2 do
      if $i==j$ then
        $\boldsymbol{b_i}[j]=B_1$
  $c_i=O_{Enc}(\boldsymbol{A},\boldsymbol{b_i},m)$
for j from 0 to $n-1$ do
  for l from 0 to 3 do
    if $c_i[j] \geqq 14$ or $c_i[j] \leqq 2$ then
      $c_i[j]=(c_i[j]+8) \bmod 16$
  if $(c_0[j] \gg 2)\&1==1$ then
    $\boldsymbol{s'}_0[j]=((c_0[j]\&7)) \| (c_3[j]\&1)-16$
  else
    $\boldsymbol{s'}_0[j]=((c_0[j]\&7)) \| (c_3[j]\&1)$
  $r_1=c_3[j]-s_0[j]$
  if $(c_1[j] \gg 2)\&1==1$ then
    $\boldsymbol{s'}_1[j]=((c_1[j]\&7)) \| ((r_1 \gg 1)\&1)-16$
  else
    $\boldsymbol{s'}_1[j]=((c_1[j]\&7)) \| ((r_1 \gg 1)\&1)$
  $r_2=r_1-2\times s_1[j]$
  if $(c_2[j] \gg 2)\&1==1$ then
    $\boldsymbol{s'}_2[j]=((c_2[j]\&7)) \| ((r_2 \gg 2)\&1)-16$
  else
    $\boldsymbol{s'}_2[j]=((c_2[j]\&7)) \| ((r_2 \gg 2)\&1)$
return $\boldsymbol{s'}$

Fig. 8. New Saber key reuse attack algorithm

Table 6. Value of c according to $\boldsymbol{s'}_0[j]$, $\boldsymbol{s'}_1[j]$, $\boldsymbol{s'}_2[j]=-3$ or -4 and $m[j]$ in Saber when $\boldsymbol{b}=[2^6,2^7,2^8]$

$\boldsymbol{s'}_0[j]$	-4	-4	-4	-4	-3	-3	-3	-3
$\boldsymbol{s'}_1[j]$	-4	-4	-3	-3	-4	-4	-3	-3
$\boldsymbol{s'}_2[j]$	-4	-3	-4	-3	-4	-3	-4	-3
$m[j]=0$	4	8	6	10	5	9	7	11
$m[j]=1$	12	0	14	2	13	1	15	3

가 존재한다(Table 6 참조).

LightSaber에서는 $\boldsymbol{b}_i=128$을 사용하게 되면 따름정리 1의 식 $\boldsymbol{b}_i \gg (\epsilon_p-\epsilon_T)=128\gg 7=1$을 만족하며 따름정리 2의 식 $(p/2-3\boldsymbol{b}_i)\gg(\epsilon_p-\epsilon_T)=(2^9-3\times 128)\gg 7=1$이 되어 $\boldsymbol{s'}_i$값의 차이가 3일 때 암호문 값이 모두 다름을 확인할 수 있다(Table 7 참조). 그리고 $\boldsymbol{b}_i=32$를 사용하면 따름정리 1의 식 $3\boldsymbol{b}_i \gg (\epsilon_p-\epsilon_T)=96\gg 7=0$을 만족하며 $4\boldsymbol{b}_i \gg (\epsilon_p-\epsilon_T)=96\gg 7=1$이고 정리 2의 ②조건을 만족한다. 즉 $\boldsymbol{s'}_i$의 값의 차이가 3 이하일 때는 암호문의 값이 같고 4 이하일 때는 암호문의 값이 다름을 알 수 있다(Table 8 참조). 이 둘을 통해 $\boldsymbol{s'}_i$를 복원할 수 있게 된다.

Saber는 $[2^5,0,0]$, $[0,2^5,0]$, $[0,0,2^5]$, $[2^6,2^7,2^8]$의 4개의 공개키를 사용하여 4번의 쿼리를 통해 $\boldsymbol{s'}_i$를 복원할 수 있고 LightSaber는 과 $[2^7,0]$, $[2^5,0]$, $[0,2^7]$, $[0,2^5]$의 4개의 공개키를 통한 4번의 쿼리로 복원할 수 있다.

결과적으로 FireSaber, Saber, LightSaber 각각에 대해 기존[9]의 4, 6, 6번의 쿼리에서 감소한 3, 4, 4번의 쿼리를 통해 $\boldsymbol{s'}_i$를 복원할 수 있었다.

Table 7. Value of c according to $\boldsymbol{s'}_0[j]$ and $m[j]$in LightSaber when $\boldsymbol{b}=[2^7,0]$

$\boldsymbol{s'}_0[j]$	-5	-4	-3	-2	-1	0	1	2	3	4	5
$m[j]=0$	3	4	5	6	7	0	1	2	3	4	5
$m[j]=1$	7	0	1	2	3	4	5	6	7	0	1

Algorithm 9. New LightSaber key reuse attack

Input : $\boldsymbol{A}$, m

$r_1=0$, $r_2=0$, $r_3=0$
$B_1=2^5$, $B_2=2^7$
for i from 0 to 3 do
  $\boldsymbol{b_i}=[0,0]$
  for j from 0 to 3 do
    $\boldsymbol{b_i}[\lfloor i/2 \rfloor]=B_{i \bmod 2}$
    $\boldsymbol{b_i}[\lfloor i/2 \rfloor]=B_{i \bmod 2}$
  $c_i=O_{Enc}(\boldsymbol{A},\boldsymbol{b_i},m)$
for j from 0 to $n-1$ do
  for l from 0 to 3 do
    if $c_i[j]\geq 6$ or $c_i[j]\leq 1$ then
      $c_i[j]=(c_i[j]+4) \bmod 8$
  if $(c_0[j]\gg 1)\&1==1$ then
    $\boldsymbol{s'}_0[j]=(c_0[j]\&3)\|(c_1[j]\&3)-16$
  else
    $\boldsymbol{s'}_0[j]=(c_0[j]\&3)\|(c_1[j]\&3)$
  if $(c_2[j]\gg 1)\&1==1$ then
    $\boldsymbol{s'}_1[j]=(c_2[j]\&3)\|(c_3[j]\&3)-16$
  else
    $\boldsymbol{s'}_1[j]=(c_2[j]\&3)\|(c_3[j]\&3)$
return $\boldsymbol{s'}$

Fig. 9. New LightSaber key reuse attack algorithm

Table 8. Value of c according to $\boldsymbol{s'}_0[j]$ and $m[j]$ in LightSaber when $\boldsymbol{b}=[2^5,0]$

$\boldsymbol{s'}_0[j]$	-5	-4	-3	-2	-1	0	1	2	3	4	5
$m[j]=0$	6	7	7	7	7	0	0	0	0	1	1
$m[j]=1$	2	3	3	3	3	4	4	4	4	5	5

4.2 한 번의 쿼리를 통한 임시 키 공격

이전의 공격들은 모두 임시 키를 여러 번 재사용해야만 공격할 수 있었으며 4.1절의 식을 통해 FireSaber, Saber, LightSaber를 각각 3, 4, 4번의 쿼리로 공격할 수 있음을 알 수 있었으며, 또한 이 횟수가 최솟값임을 알 수 있었다. 이는 그 이하로 임시 키를 재사용하면 공격이 쉽지 않음을 의미하기도 한다. 본 절에서는 매우 적은 횟수로 임시 키를 재사용할 때, 이에 대한 공격을 위해 한 번의 쿼리로 임시 키를 복원하는 데 필요한 계산 복잡도를 줄이는 방법을 설명한다.

임시 키 $\boldsymbol{s'}_i$에 대한 전수조사 복잡도는 이항분포(CBD) 및 격자의 차수 n에 의해 $(\mu+1)^{kn}$이 된다. 이때 암호문 $c[j]$값에 대응되는 임시 키 $\boldsymbol{s'}_i[j]$값을 균등하게 분배할 수 있는 공개키가 있다고 가정하자. 암호문의 계수 값이 가질 수 있는 경우의 수는 T이기 때문에 암호문의 한 계수 값에 대해 대응될 수 있는 $\boldsymbol{s'}_i[j]$값의 개수는 $\{(\mu+1)^k\}/T$이다. n차 다항식을 사용하며 평문 값에 따라서도 암호문의 값이 변하므로 전체 $\boldsymbol{s'}_i$값을 구하는 데 필요한 계산 복잡도는 $[\{2\times(\mu+1)^k\}/TRIGHT]^n$이 되며 이때 $(\mu+1)^k$개의 테이블이 있어야 한다. $T>2$이므로 이전의 전수조사 복잡도보다 확실히 작아짐을 알 수 있으며 다음의 따름정리를 통해 우리가 원하는 공개키를 얻을 수 있다.

따름정리 3. $\boldsymbol{s}_{i,Y}(=m\cdot p/2)\in R_q$이고 $\boldsymbol{s}_i$의 계수들이 $\boldsymbol{s}_i[j]\in\{-\mu/2,\cdots,\mu/2\}$, $\mu/2<p$를 만족하고 Y의 계수가 $Y[j]\in\{0,p/2RIGHT,i=1,\cdots,n$ 이다. 공개키의 각 원소에 대해 상수항을 제외한 나머지 항들의 계수가 0의 값을 가져 $\boldsymbol{b}_i\in\mathbb{Z}_p$의 형태를 가지고 있으며 $V=\left(\sum_{i=0}^{k-1}\boldsymbol{b}_i\boldsymbol{s}_i[j]\right)+h_1-Y \bmod p$라고 하자. 그리고 암호문은 $c=V\gg(\epsilon_p-\epsilon_T)$과 같이 계산된다.

이때, $\left(\sum_{i=0}^{k-1}r_i\boldsymbol{b}_i\right)\gg(\epsilon_p-\epsilon_T)=1$을 만족하면 각 $\boldsymbol{s}_i[j]$들의 값의 차이가 r_i일 때 암호문 값은 1의 차이가 난다.

LightSaber에서 각 암호문에 가능한 $\boldsymbol{s'}_i[j]$ 값이 균등하게 분포되려면 $\boldsymbol{s'}_0[j]$ 값이 같고 $\boldsymbol{s'}_1[j]$의 값의 차이가 1이 날 때 평문이 같은 암호문의 값 또한 1의 차이가 나야 하며, $\boldsymbol{s'}_0[j]$값의 차이가 1일 때는 $\boldsymbol{s'}_1[j]$의 값이 10만큼 감소할 때 평문이 같은 암호문의 값이 1의 차이를 가져야 한다. 이는 따름정리 3에 의해 다음 두 식

$$\boldsymbol{b}_1\gg 7=1 \tag{1}$$

$$(\boldsymbol{b}_0-10\boldsymbol{b}_1)\gg 7=1 \tag{2}$$

을 만족하는 공개키가 우리가 원하는 공개키가 되고 이를 만족하는 값은 $\boldsymbol{b}_0=384$, $\boldsymbol{b}_1=128$이다.

Saber에서도 마찬가지로 $\boldsymbol{s'}_0[j]$와 $\boldsymbol{s'}_1[j]$값이 같고 $\boldsymbol{s'}_2[j]$의 값의 차이가 1이 날 때 평문이 같은 암호문의 값 또한 1의 차이가 나야 하며 $\boldsymbol{s'}_1[j]$값의 차이가 1일 때는 $\boldsymbol{s'}_2[j]$값이 8만큼 감소할 때 평문이 같은 암호문의 값이 1의 차이를 가져야 한다. 그리고 $\boldsymbol{s'}_2[j]$값의 차이가 1일 때는 $\boldsymbol{s'}_1[j]$, $\boldsymbol{s'}_2[j]$ 모두 8만큼 감소할 때 평문이 같은 암호문의 값이 1의 차이를 가져야 한다. 이는 따름정리 3에 의해 다음 3개의 식

$$\boldsymbol{b}_2\gg 6=1 \tag{3}$$

$$(\boldsymbol{b}_1-8\boldsymbol{b}_2)\gg 6=1 \tag{4}$$

$$(\boldsymbol{b}_0-8\boldsymbol{b}_1-8\boldsymbol{b}_2)\gg 6=1 \tag{5}$$

을 만족하는 공개키가 우리가 원하는 공개키가 되고 이를 만족하는 값은 $\boldsymbol{b}_0=64$, $\boldsymbol{b}_1=576$, $\boldsymbol{b}_2=64$이다.

마지막으로 FireSaber에서도 LightSaber 및 Saber와 비슷한 과정을 통해 따름정리 3에 의해 다음과 같은 4개의 식

$$\boldsymbol{b}_3\gg 4=1 \tag{6}$$

$$(\boldsymbol{b}_1-6\boldsymbol{b}_2)\gg 4=1 \tag{7}$$

$$(\boldsymbol{b}_0-6\boldsymbol{b}_1-6\boldsymbol{b}_2)\gg 4=1 \tag{8}$$

$$(\boldsymbol{b}_0-6\boldsymbol{b}_1-6\boldsymbol{b}_2-6\boldsymbol{b}_3)\gg 4=1 \tag{9}$$

을 만족하는 공개키가 우리가 원하는 공개키가 되고 이를 만족하는 값은 $\boldsymbol{b}_0=368$, $\boldsymbol{b}_1=784$, $\boldsymbol{b}_2=112$, $\boldsymbol{b}_3=16$이다.

LightSaber, Saber, FireSaber에서 각각 $[384,128]$, $[64,576,64]$, $[368,784,112,16]$을 사용하여 실제 실험을 진행하게 되면 하나의 $c[j]$와 대응될 수 있는 $\boldsymbol{s'}_i[j]$의 값이 각각 $2^{4.91}$, $2^{6.5}$, $2^{6.22}$개가 된다. 즉, $\boldsymbol{s'}_i$를 복원하는 계산 복잡도가 $2^{4.91\times n}$, $2^{6.5\times n}$, $2^{6.22\times n}$이 되는 것으로 원래의 값인 $2^{7.91\times n}$, $2^{10.51\times n}$, $2^{12.22\times n}$에 비해 제곱근에 가깝게 감소시킬 수 있었다.

V. 결 론

본 공격은 기본적으로 매우 특수한 형태의 공개키를 사용하여야 한다는 것이 가장 큰 한계점으로 다가온다. 실제로 알고리즘이 실행되게 되면 상수항을 제외한 나머지 계수가 0일 확률이 극히 낮으며 그중에서도 특수한 몇 개의 값이 선택되어야 하므로 자연적으로 공격에서 사용되는 공개키가 선택될 가능성이 매우 낮다. 그래서 이 공격은 다음과 같은 방법을 통해 진행될 수 있다. 사용자와 서버가 통신을 진행할 때 공격자가 통신 과정에서 중간자 공격을 통해 서버의 공개키가 아닌 악의적으로 선택된 공개키를 전송함으로써 사용자가 재사용하는 임시 키를 알아낼 수 있다. 그렇게 임시 키를 복원할 수 있게 되면 통신에서 사용된 평문을 쉽게 복원할 수 있어서 위험할 수 있다.

4.1절의 공격 방법은 각 파라미터 별로 3, 4, 4번의 쿼리를 통해 이루어지며, 4.2절의 공격 방법 또한 결국 평문을 복원하는 계산 복잡도인 2^n보다 높아서 해당 공격에 대해 가장 낮은 안전 강도를 가지는 LightSaber에서도 4번 이상 재사용하지 않으면 평문을 복원하는 것보다 비효율적이다. 위와 같은 상황을 고려해보았을 때, 공격의 대응. 방안으로써 특수한 형태의 공개키가 사용되지 못하도록 제한하거나 임시 키의 재사용 횟수가 일정 이상 되지 못하도록 제한하는 방법이 있다. 이러한 방법들을 사용하여 통신 간의 임시 키 재사용을 안전하게 할 수 있다.

[9]에서는 CRYSTALS-KYBER[2]에 대한 임시 키 재사용 공격 또한 제시되었다. CRYSTALS-KYBER는 마찬가지로 Meta-PKE 구조를 가지고 있으며 암호화 과정에서 사용되는 압축 함수가 modulo 구조로 되어 있어서 SABER 알고리즘과 비슷하다. 암호화 과정에서 에러의 영향을 제거할 수 있다면 압축 함수 이전의 경우의 수가 이후의 것보다 작아지는 쿼리 수를 찾을 수 있다. 이 쿼리 수가 [9]에서 제시된 것보다 더 적기 때문에 본 논문에서 제시한 공격 방법과 비슷하게 적용하여 공격을 향상시킬 수 있을 것으로 기대하며 이에 대한 연구를 진행할 예정이다.

References

[1] P.W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," SIAM Journal Computing, vol. 26, no. 5, pp. 1484-1509, Oct. 1997.

[2] J. Bos et al., "CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM," 2018 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 353-367, Apr. 2018.

[3] J.P. D'Anvers, A. Karmakar, S.S. Roy, and F. Vercauteren, "Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM," Progress in Cryptology -AFRICACRYPT 2018, AFRICACRYPT 2018, pp. 282-305, Apr. 2018.

[4] E. Rescorla, "The Transport Layer Security (TLS) protocol version 1.3," RFC 8446, Aug. 2018.

[5] E. Alkim, L. Ducas, T. Pöppelmann, P. Schwabe, "Post-quantum key exchange - a new hope," 25th Usenix Security Symposium, USENIX Association, pp. 327-343, Aug. 2016.

[6] C. Liu, Z. Zheng, G. Zou, "Key Reuse Attack on NewHope Key Exchange Protocol," Information Security and Cryptology - ICISC 2018, ICISC 2018, pp. 163-176, Jan. 2019.

[7] K. Wang, Z. Zhang, and H. Jiang, "Security of two NIST candidates in the presence of randomness reuse," Provable and Practical Security, ProvSec 2020, pp. 402-421, Nov. 2020.

[8] X. Lu et al., "LAC: Practical Ring-LWE Based Public-Key Encryption with Byte-Level Modulus," IACR ePrint 2018-1009, Oct. 2018.

[9] S. Okada, Y. Wang, "Recovery Attack on Bob's Reused Randomness in CRYSTALS-KYBER and SABER," Provable and Practical Security, ProvSec 2021, pp. 155-173, Nov. 2021.

〈저 자 소 개〉

이 창 원 (Changwon Lee) 학생회원
2021년 2월: 서울시립대학교 수학과 졸업
2021년 3월~현재: 고려대학교 정보보호대학원 석사과정
〈관심분야〉 후양자암호

전 찬 호 (Chanho Jeon) 학생회원
2019년 2월: 고려대학교 수학과 졸업
2019년 3월~현재: 고려대학교 정보보호대학원 석박사 통합과정
〈관심분야〉 후양자암호

김 수 리 (Suhri Kim) 정회원
2014년 2월: 고려대학교 수학과 이학사
2016년 8월: 고려대학교 정보보호학과 공학석사
2020년 2월: 고려대학교 정보보호대학원 공학박사
2020년 3월~2021년 2월: 고려대학교 정보보호대학원 박사후연구원
2020년 3월~2021년 2월: K.U. Leuven ESAT/SCD-COSIC 박사후 연구원
2022년 3월~현재: 성신여자대학교 수리통계데이터사이언스학부 조교수
〈관심분야〉 공개키 암호시스템, 후양자암호

홍 석 희 (Seokhie Hong) 종신회원
1995년: 고려대학교 수학과 학사
1997년: 고려대학교 수학과 석사
2001년: 고려대학교 수학과 박사
1999년 8월~2004년 2월: ㈜시큐리티 테크놀로지 선임연구원
2003년 3월~2004년 2월: 고려대학교 정보보호기술연구센터 선임연구원
2004년 4월~2005년 2월: K.U. Leuven ESAT/SCD-COSIC 박사후연구원
2005년 3월~2013년 8월: 고려대학교 정보보호대학원 부교수
2013년 9월~현재: 고려대학교 정보보호대학원 정교수
〈관심분야〉 대칭키 및 공개키 암호 알고리즘, 부채널 공격 및 대응기법, 디지털포렌식