정 보 보 호 학 회 지
제 32 권 제2호, 2022. 4

사물인터넷 장비 상에서의 양자내성암호 구현 동향

권 혁 동*, 엄 시 우*, 심 민 주*, 서 화 정**

요 약

양자컴퓨터의 발전에 따라 양자알고리즘에 대한 보안성을 보장하는 양자내성암호의 중요성이 대두되고 있다. 미국 국립표준기술연구소 (NIST, National Institute of Standards and Technology)는 양자내성암호 표준화 공모전을 개최하여 차세대 공개키 암호에 대한 검증을 현재 Round 3까지 진행한 상태이다. 본 고에서는 최근에 활발히 연구되고 있는 NIST 양자내성암호 공모전에 제안된 양자내성암호에 대해서 확인해 보도록 한다. 또한 해당 양자내성암호의 연산 효율성을 높이기 위해 사물인터넷 장비 상에서 최적 구현 기법에 대해 확인해 보도록 한다.

Ⅰ. 서 론

양자컴퓨터의 발전에 따라 기존 공개키 암호 알고리즘에 대한 위협이 증가하고 있다. 이를 해결하기 위해 양자내성암호에 대한 필요성이 증가하고 있으며 이에 대응하여 미국 국립표준기술연구소(NIST, National Institute of Standards and Technology)는 양자내성암호 표준화를 위한 공모전을 개최하여 진행 중에 있다. 본 고에서는 양자컴퓨터의 발전과 양자내성암호의 최신 동향에 대해서 확인해 보도록 한다. 또한 한정된 자원 환경을 가지는 사물인터넷 기기 상에서의 양자내성암호 최적화 구현 기법에 대해서 확인해 보도록 한다.

Ⅱ. 양자컴퓨터의 도래와 양자내성암호 공모전

양자컴퓨터는 양자역학을 이용한 컴퓨터로 Feynman의 아이디어에서 비롯되었다[1]. 일반적인 컴퓨터가 전기 신호를 사용하여 0과 1을 표현한다면, 양자컴퓨터는 0과 1의 중첩 상태를 사용한다. 양자컴퓨터는 중첩상태를 표현하기 위해서 큐빗 (qubit)이라는 표기법을 사용하며, 지속적인 연구로 양자컴퓨터 상에서 운용가능한 큐빗의 수는 점차 증가하고 있다. 2019년 구글에서 발표한 Sycamore 프로세서는 54개의 큐빗을 가지고 있었지만 [2], 2021년 IBM에서 발표한 Eagle 프로세서는 127 큐빗을 보유하고 있다 [3]. 현재 IBM은 표 1과 같은 로드맵을 공개하며 양자컴퓨터의 전망을 보여주고 있다.

양자컴퓨터의 발전은 기술 발전 측면에서 이로운 점이 많지만 암호 분야에서는 큰 위협으로 다가오고 있다. 현대 암호의 안전성은 수학적 문제의 복잡함에 기반한다. 하지만 양자알고리즘은 일부 수학 문제를 손쉽게 풀 수 있다. Grover가 제안한 Grover 알고리즘은 n개의 데이터 상에서 특정 데이터를 찾아내는 알고리즘이다. 기존 Brute force attack은 $O(2^n)$의 검색 횟수가 필요하다면, Grover 알고리즘을 사용한 경우에는 최대 $O(2^{n/2})$ 검색으로 데이터를 찾아낼 수 있다. 이는 대칭키 알고리즘과 해시함수에 특히 위협적이다 [4]. Shor 알고리즘은 소인수 분해에 최적화된 알고리즘으로, 기

〔표 1〕 IBM 양자컴퓨터 로드맵.

Year	Qubit	Processor
2019	27	Falcon
2020	65	Hummingbird
2021	127	Eagle
2022	433	Osprey
2023	1,121	Condor
2030	1million	-

본 연구는 2022년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. NRF-2020R1F1A1048478, 100%).
* 한성대학교 IT융합공학부 (대학원생, korlethean@gmail.com, 대학원생, shuraatum@gmail.com, 대학원생, minjoos9797@gmail.com)
** 한성대학교 IT융합공학부 (조교수, hwajeong84@gmail.com)

존 *n*-bit의 숫자를 소인수 분해할 때는 지수함수적으로 연산량이 증가하나 Shor 알고리즘을 사용할 경우 연산량이 로그함수적으로 증가하게 된다 [5]. 따라서 소인수 분해를 중점으로 사용하는 공개키 알고리즘은 Shor 알고리즘에 위협을 받게 된다.

미국의 NIST는 현재 암호체계는 양자컴퓨터 시대에 붕괴될 것으로 판단하여, 양자내성암호 표준화 공모전을 개최하여 운영 중에 있다. Round 3까지 진행한 결과, 최종후보군에는 공개키 알고리즘 4개, 전자서명 알고리즘 3개가 진출하였으며 대체후보군에는 공개키 알고리즘 5개, 전자서명 알고리즘 3개가 진출하였다.

Ⅲ. 양자내성암호 소개

본 장에서는 NIST 양자내성암호 공모전의 일부 양자내성암호에 대해서 소개한다.

3.1. Saber

Saber는 NIST 양자내성암호 공모전의 Round 3에 진출한 공개키 알고리즘으로, 격자 기반 암호 중 하나이다 [6]. Saber는 Module Learning With Rounding (M-LWR)을 사용한다. LWR은 Learning With Errors (LWE)와 비슷한 문제로, LWE는 연산 결과에 오류를 주입하는 반면 LWR은 연산 결과에 Least Significant Bit (LSB) 일부를 버린다. Saber는 수식 1의 ring 상에서 연산을 진행한다. 수식 2는 ring의 매개변수이다.

$$R_q = Z_q[x]/ < x^{256} + 1 > \tag{1}$$

$$q = 2^{13},\ n = 256 \tag{2}$$

표 2는 Saber에서 사용되는 매개변수를 나열한 것이다. Saber의 종류별로 각기 다른 매개변수를 확인할 수 있다.

Saber는 Fujisaki-Okamoto 변환을 응용하여 안전성을 확보하였고, *2n*의 modulus를 사용하는 것으로 modular reduction 연산을 최적화하였다. 이는 Number Theoretic Transform (NTT)를 사용하지 않고서도 빠른 속도로 곱셈 연산을 할 수 있다. 표 3, 4, 5는 CPA 보안을 제공하는 키 생성, 암호화, 복호화 과정을 각각 나타내고 있다 [7].

〔표 2〕 Saber의 파라미터, Type: 알고리즘 이름, *l*: 모듈 디멘션, T: 라운딩 파라미터, μ: 비밀 분산 값 파라미터.

Type	*l*	$T = 2^{\in T}$	μ
Lightsaber	2	2^3	10
Saber	3	2^4	8
Firesaber	4	2^6	6

〔표 3〕 Saber 키 생성 과정.

Output: $pk = (seed_A, b)$, $sk = (s)$
1: $seed_A \leftarrow Sample_U()$
2: $A \in R_q^{l \times l} \leftarrow Expand(seed_A)$
3: $s \in R_q^{l} \leftarrow Sample_B()$
4: $b \leftarrow Round(A^T \cdot s)$

〔표 4〕 Saber CPA 암호화 과정.

Input: m, r, $pk = (seed_A, b)$
Output: $ct = (c, b')$
1: $A \in R_q^{l \times l} \leftarrow Expand(seed_A)$
2: $s' \in R_q^{l} \leftarrow Sample_B(r)$
3: $b' \leftarrow Round(As')$
4: $v' \leftarrow b^T(s' \bmod p)$
5: $c \leftarrow Round(v' - 2^{\in -1} m)$

〔표 5〕 Saber CPA 복호화 과정.

Input: $ct = (c, b')$, $sk = (s)$
Output: m
1: $v' \leftarrow b'^T(s \bmod p)$
2: $m \leftarrow Round(v - 2^{\in p - \in T} c \bmod p)$

Saber의 키 생성과 암호화에서 가장 많은 시간이 소요되는 작업은 다항식 행렬 곱셈으로, AT·s, As'이다. 복호화 과정에서는 b'T·s 내적 연산에 많은 연산 자원이 소모된다.

3.2. Dilithium

Dilithium은 NIST 양자내성암호 공모전의 Round 3에 진출한 전자서명 알고리즘으로 격자 기반 문제를

사용한다 [8]. Dilithium은 Module Small Integer Solutions (M-SIS)와 M-LWE 문제에 기반한다. Dilithium에서 사용하는 모듈은 수식 3과 같으며, 수식 4는 Dilithium의 ring의 매개변수이다.

$$R_q = Z_q[x]/ < x^{256} + 1 > \tag{3}$$

$$q = 2^{23} - 2^{13} + 1 = 8380417 \tag{4}$$

표 6은 Dilithium에서 사용되는 매개변수의 종류를 나열한다. 보안 강도 별로 Dilithium의 종류가 나누어져 있으며, 각각의 Dilithium 마다 사용하는 매개변수를 확인할 수 있다.

Dilithium의 키 생성, 서명 생성 및 검증에서 ($k \times l$) × ($l \times 1$) 행렬-벡터 다항식 곱셈이 주된 연산을 차지한다. 이 연산은 반복적으로 실행되며 연산 시간의 대부분을 차지한다. 추가적으로 SHAKE-256을 사용하여 연산 효율을 향상시켰다.

3.3. Kyber

Kyber는 Dilithium과 기본적인 특성과 구조를 공유하는 알고리즘으로, NIST 양자내성암호 공모전 Round 3에 진출하였다 [8]. Kyber는 공개키 알고리즘으로 Dilithium과 동일하게 격자 기반 문제 중 하나인 M-LWE를 기반으로 한다. Kyber는 Dilithium과 같은 ring을 사용하나, 수식 5의 매개변수를 사용한다.

$$q = 3329,\ n = 256 \tag{5}$$

Kyber는 Fujisaki-Okamoto 변환을 사용하며 cyclotomic ring을 적용하는 것으로 NTT를 사용할 수 있다. 표 7은 Kyber의 매개변수를 나열한 것이다. 대부분의 매개변수의 크기가 작기 때문에 효율적인 연산 및 관리가 가능하다.

〔표 7〕 Kyber의 파라미터, Type: 알고리즘 이름, /: 모듈 디멘션, T: 라운딩 파라미터, μ: 비밀 분산 값 파라미터.

Type	l	(d1,d2)	η(s|s')	η(e|e'|e")
Kyber512	2	(10,4)	6	4
Kyber768	3	(10,4)	4	4
Kyber1024	4	(11,5)	4	4

3.4. SIKE

SIKE는 Supersingular Isogeny Key Encapsulation의 줄임말로, NIST 양자내성암호 공모전의 Round 3 대체후보군에 진출한 공개키 알고리즘이다. SIKE는 SIDH (Supersingular Isogeny Diffie-Hellman)를 KEM 버전으로 확장한 암호로서, 양자내성암호 공모전에 남은 암호 중 유일하게 타원 곡선 기반의 아이소지니 (isogeny) 문제에 기반한다[9]. SIKE는 키와 암호문의 크기가 매우 작다는 장점을 갖지만 연산속도가 매우 느리다는 단점이 있다[10]. 표 8, 9, 10은 SIKE의 키 생성, 암호화, 복호화 과정을 각각 나타내고 있다[11].

〔표 8〕 SIKE 키 생성 과정.

Output: pk_3, sk_3, s
1: $(sk_3, pk_3) \leftarrow Gen()$
2: $s \leftarrow_{\$} \{0,1\}^n$
3: $b \leftarrow Round(A^T \cdot s)$

〔표 9〕 SIKE 암호화 과정.

Output: k, (c_1, c_2)
1: $m \leftarrow_{\$} \{0,1\}^n$
2: $sk_2 \leftarrow SHAKE256(m \,||\, pk_3)$
3: $b \leftarrow Round(A^T \cdot s)$
4: $(c_1, c_2) \leftarrow Enc(pk_3, m, sk_2)$
5: $k \leftarrow SHAKE256(m \,||\, (c_1, c_2))$

〔표 6〕 Dilithium의 파라미터, Type: 알고리즘 이름, NIST level: NIST 권장 보안 강도, (k, l): 행렬 크기, η: 샘플링 경계 값, β, ω: 거부 임계 값, $|pk|$: 공개키 크기(byte), $|sig|$: 서명 크기(byte), exp. 반복 횟수.

Type	NIST level	(k, l)	η	β	ω	$\|pk\|$	$\|sig\|$	exp. iterations
Dilithium2	2	(4, 4)	2	78	80	1312	2420	4.25
Dilithium3	3	(6, 5)	4	196	55	1952	3293	5.1
Dilithium4	5	(8, 7)	2	120	75	2592	4595	3.85

〔표 10〕 SIKE 복호화 과정.

Output: k	
1:	$m' \leftarrow Dec(sk_3, (c_1, c_2))$
2:	$sk_2 \leftarrow SHAKE256(m' \| \| pk_3)$
3:	$x_{R2} \leftarrow x_{P2} + [sk'_2]x_{Q2}$
4:	$(\Phi_2, E_2) \leftarrow isogeny_2(E_0, x_{R2})$
5:	$c'1 \leftarrow (\Phi_2(x_{P3}), \Phi_2(x_{Q3}), \Phi_2(x_{PQ3}))$
6:	**if** $c'_1 == c_1$ **then**
7:	$k \leftarrow SHAKE256(m' \| \| c_1, c_2))$
8:	**else**
9:	$k \leftarrow SHAKE256(s \| \| c_1, c_2))$
10:	**endif**

3.5. Rainbow

Rainbow는 NIST 양자내성암호 공모전의 Round 3 대체후보군 알고리즘 중 하나로, 다변수 기반의 전자서명 알고리즘이다. Rainbow는 UoV(Unbalanced Oil-Vinegar) 문제를 사용하여 빠른 속도로 서명을 생성 및 검증이 가능하며, 다른 전자서명 알고리즘에 비해 짧은 서명 길이를 가진다[12]. Rainbow는 일반적인 버전인 Standard와 키 사이즈를 줄인 Cyclic의 두 가지가 존재한다. 표 11은 Rainbow의 매개변수를 소개한다[13].

표 12, 13, 14는 Rainbow 전자서명의 키 생성, 서명 생성, 검증 과정을 나타낸 것이다[14].

〔표 11〕 Rainbow의 파라미터. Lv: NIST 권장 보안 강도, $|pk|$: 공개키 크기(KB), $|sk|$: 비밀키 크기(KB), $|sig|$: 서명 크기(KB).

Standard				
Lv	Parameters	$\|pk\|$	$\|sk\|$	$\|sig\|$
I	(GF(16), 36, 32, 32)	157.8	101.2	528
III	(GF(256), 68, 32, 48)	861.4	611.3	1,312
V	(GF(256), 96, 32, 64)	1,885.4	1,375.7	1,632
Cyclic				
Lv	Parameters	$\|pk\|$	$\|sk\|$	$\|sig\|$
I	(GF(16), 36, 32, 32)	58.8	101.2	528
III	(GF(256), 68, 32, 48)	258.4	611.3	1,312
V	(GF(256), 96, 32, 64)	523.5	1,375.7	1,696

〔표 12〕 Rainbow 키 생성 과정.

Input: q, v_1, o_1, o_2, l	
Output: public key(pk), secret key(sk)	
1:	$m \leftarrow o_1 + o_2$
2:	$n \leftarrow m + v_1$
3:	**for** IsInvertible(M_S) == True
4:	$M_S \leftarrow Matrix(q, m, m)$
5:	**endfor**
6:	$\mathcal{S} \leftarrow M_S$
7:	$Inv\mathcal{S} \leftarrow M_S^{-1}$
8:	**for** IsInvertible(M_T) == True
9:	$M_T \leftarrow Matrix(q, n, n)$
10:	**endfor**
11:	$\mathcal{T} \leftarrow M_T$
12:	$Inv\mathcal{T} \leftarrow M_T^{-1}$
13:	$\mathcal{F} \leftarrow Rainbowmap(q, v_1, o_1, o_2)$
14:	$\mathcal{P} \leftarrow \mathcal{S} \circ \mathcal{F} \circ \mathcal{T}$
15:	$sk \leftarrow (Inv\mathcal{S}, \mathcal{F}, Inv\mathcal{T}, l)$
16:	$pk \leftarrow (\mathcal{P}, l)$

〔표 13〕 Rainbow 서명 생성 과정.

Input: d, $pk(Inv\mathcal{S}, \mathcal{F}, Inv\mathcal{T})$, l	
Output: signature σ = (z, r)	
1:	**for** IsInvertible($\hat{F}$) == True
2:	$y_1, \ldots, y_{v1} \leftarrow {}_R\mathbb{F}_q$
3:	$\hat{f}^{(v_1+1)}, \ldots, \hat{f^{(n)}}$ $\leftarrow f^{(v_1+1)}(y_1, \ldots, y_{v1}), \ldots, f^{(n)}(y_1, \ldots, y_{v1})$
4:	$(\hat{F}, C_F) \leftarrow Aff^{-1}(f^{(v_1+1)}, \ldots, \hat{f}^{(n)})$
5:	**endfor**
6:	$Inv\mathcal{F} \leftarrow \hat{F}^{-1}$
7:	**for** t == True
8:	$r \leftarrow \{0, 1\}^l$
9:	$h \leftarrow \mathcal{H}(\mathcal{H}(d)\|\|r)$
10:	$x \leftarrow Inv\mathcal{S} \cdot h$
11:	$(y_{v_1+1}, \ldots, y_{v2}) \leftarrow Inv\mathcal{F} \cdot ((x_{v1+1}, \ldots, x_{v2}) - C_F)$
12:	$\hat{f}^{(v_2+1)}, \ldots, \hat{f}^{(n)}$ $\leftarrow \hat{f}^{(v_2+1)}(y_{v_1+1}, \ldots, y_{v2}), \ldots, \hat{f}^{(n)}(y_{v_1+1}, \ldots, y_{v2})$
13:	$t, ((y_{v_2+1}, \ldots, y_n))$ $\leftarrow Gauss(\hat{f}^{(v_2+1)} = x_{v_2+1}, \ldots, \hat{f}^{(n)} = x_n)$
14:	**endfor**
15:	$z = Inv\mathcal{T} \cdot y$
16:	$\sigma \leftarrow (z, r)$

〔표 14〕 Rainbow 서명 검증 과정.

Input: d, signature σ = (z, r)
Output: boolean value True or False
h ← $\mathcal{H}(\mathcal{H}(d)\|\|r)$
h' ← $\mathcal{P}(z)$
if h' == h **then**
return True
else
return False
endif

3.6. UoV(Unbalanced Oil-Vinegar)

UoV의 원형인 Oil and Vinegar는 Jacques Patarin이 디자인한 알고리즘으로, 다변수 기반의 전자서명 알고리즘이다. Oil and Vinegar는 단순한 구조를 지니고 빠른 연산속도를 제공하며 RAM 소모량이 적기 때문에 스마트카드와 같은 자원이 제한된 환경에서 원활하게 구현되었다 [15]. Oil and Vinegar는 선형 비밀함수를 사용하여 유한체 K상에 Oil 값 n개와 $v = n$개인 Vinegar에 2차 방정식을 숨기게 된다. 하지만 Kipnis와 Shamir가 Oil and Vinegar 구조의 취약점을 발견하였다 [16].

이후 Oil and Vinegar를 개선한 UoV가 제안되었다. UoV는 $v = n$ 대신 $v > n$의 비대칭 형태로 간단한 변형을 주었다. Oil 값 보다 Vinegar 값이 더 많기 때문에 UoV라는 이름을 갖게 되었다[17]. UoV는 Oil and Vinegar과 동일한 다변수 기반의 알고리즘이다. UoV는 2차 방정식을 풀기 위해 n개의 변수로 m개의 방정식을 푸는 것이 NP-hard라는 점에 착안하였다 [18].

표 15, 16, 17는 UoV 전자서명의 키 생성, 서명 생성, 검증 과정을 나타낸 것이다[14].

〔표 15〕 UoV 키 생성 과정.

Input: q, v, o, l
Output: public key(pk), secret key(sk)
m ← o
n ← m + v
for IsInvertible(M_T) == True
M_T ← Matrix(q, n, n)
endfor
$\mathcal{T} \leftarrow M_T$
$Inv\mathcal{T} \leftarrow M_T^{-1}$
$\mathcal{F}$ ← UoVmap(q, v, o)
$\mathcal{P} \leftarrow \mathcal{F} \circ \mathcal{T}$
$sk \leftarrow (\mathcal{F}, Inv\mathcal{T}, l)$
$pk \leftarrow (\mathcal{P}, l)$

〔표 16〕 UoV 서명 생성 과정.

Input: d, $pk(\mathcal{F}, Inv\mathcal{T})$, l
Output: signature σ = (z, r)
for IsInvertible($\hat{F}$) == True
$y_1,\ldots,y_v \leftarrow {}_R\mathbb{F}_q$
$\hat{f}^{(v_1+1)},\ldots,\widehat{f^{(n)}}$ $\leftarrow f^{(v+1)}(y_1,\ldots,y_v),\ldots,f^{(n)}(y,\ldots,y_v)$
$(\hat{F}, C_F) \leftarrow \mathrm{Aff}^{-1}(f^{(v_1+1)},\ldots,\hat{f}^{(n)})$
endfor
$Inv\mathcal{F} \leftarrow \hat{F}^{-1}$
r ← $\{0, 1\}^l$
x ← $\mathcal{H}(\mathcal{H}(d)\|\|r)$
$(y_{v_1+1},\ldots,y_n) \leftarrow Inv\mathcal{F}\cdot((x_{v+1},\ldots,x_n)-C_F)$
z = $Inv\mathcal{T}\cdot y$
$\sigma \leftarrow$ (z, r)

〔표 17〕 UoV 서명 검증 과정.

Input: d, signature σ = (z, r)
Output: boolean value True or False
h ← $\mathcal{H}(\mathcal{H}(d)\|\|r)$
h' ← $\mathcal{P}(z)$
if h' == h **then**
return True
else
return False
endif

3.7. ROLLO

ROLLO (Rank-Ouroboros, Lake and LOcker)는 코드 기반 암호로 고전 컴퓨터와 양자 컴퓨터 양쪽 모두에서 안전하게 사용할 수 있도록 구현된 양자내성 공개키 암호 알고리즘이다[19]. ROLLO는 NIST 양자내

성암호 공모전의 Round 2 진출 암호로, 일반적인 코드 기반 암호가 Goppa 코드를 사용하지만 ROLLO는 Rank metric [20] 코드를 사용하여 키 사이즈를 줄이고 연산 효율을 늘렸다. 표 18은 ROLLO의 매개변수를 나타낸 것이다[21].

〔표 18〕 ROLLO의 파라미터.

Type	n	m	r	d	Lv
ROLLO-I-128	83	67	7	8	1
ROLLO-I-192	97	79	8	8	3
ROLLO-I-256	113	97	9	9	5
ROLLO-II-128	189	83	7	8	1
ROLLO-II-192	192	97	8	8	3
ROLLO-II-256	211	97	9	9	5
ROLLO$^+$-I-128	83	67	7	8	1
ROLLO$^+$-I-192	97	79	8	8	3
ROLLO$^+$-I-256	113	97	9	9	5
ROLLO$^+$-II-128	189	83	7	8	1
ROLLO$^+$-II-192	192	97	8	8	3
ROLLO$^+$-II-256	211	97	9	9	5

Ⅳ. 사물인터넷 프로세서 소개

4.1. 저성능 사물인터넷 프로세서

Cortex-M 프로세서는 저성능 프로세서로 낮은 비용과 낮은 전력 소모로 다양한 확장성을 제공하며, IoT에서 매우 효율적인 특징을 가지고 있다. Cortex-M 프로세서는 ARM 기반 프로세서로 RISC (reduced in-struction set computer) 형태의 낮은 전력, 높은 성능의 임베디드 프로세서로 시장의 95%를 점유하고 있다 [22].

4.1.1. Cortex-M3

Arduino DUE의 프로세서로 활용되는 Cortex-M3 프로세서는 IoT 구현에 매우 효율적인 특징을 가진다. Cortex-M3의 장점으로는 32-bit로 동작하는 연산자, 낮은 전력 프로세싱 그리고 저렴한 가격이 있다. Cortex-M3 프로세서는 3단계 파이프라인을 지원하며, ARMv7-M 구조를 따른다. Thumb-1과 Thumb-2 명령어 셋을 지원하며, Thumb-1의 경우 16-bit 단위로 연산자를 구성하는데 이를 이용하면 동일한 32-bit ARM 연산자로 동작하는 코드를 절반으로 줄이는 구현이 가능하다 [23].

4.1.2. Cortex-M4

Cortex-M4는 NIST에 의해 공모전 프로세스 상에서 후보군들을 평가하기 위한 표준 임베디드 플랫폼으로 추천되었다. Cortex-M4 프로세서는 ARMv7E-M 구조를 따르는 32-bit 프로세서이며, 핵심 특징은 다음과 같다.

Cortex-M4는 14개의 범용 레지스터를 가진다. r0-r15의 16개 레지스터를 가지나, 스택 포인터 (r13)과 프로그램 카운터 (r15)를 제외한 14개의 레지스터를 범용 레지스터로 사용가능하다. Store 명령어는 항상 1 cycle이며, 연관 없는 H개의 배열을 Load할 때는 (H+1) cycle 이다. Cortex-M3보다 효율적인 명령어를 지원하여 긴 곱셈과 누적 곱셈을 낮은 비용으로 연산이 가능하다. 배럴 시프터를 지원하여, 추가 비용 없이 시프트 및 회전 연산을 구현 가능하다 [6].

4.2. 고성능 사물인터넷 프로세서

4.2.1. Cortex-A72

ARM이 2015년에 발표한 Cortex-A72의 경우 동일한 프로세서 설계에서 전력 감소와 성능 향상이 결합되어 효율성이 크게 향상되었다. Cortex-A72는 스마트폰 전력 예산에서 3.5배 향상된 성능을 제공할 수 있다. Cortex-A72는 ARMv8-A 구조를 따르며 64-bit를 지원하면서 기존 32-bit 소프트웨어에 대한 하위 호환성도 지원한다. Cortex-A72는 모바일, 네트워킹 및 인프라/데이터 센터에서 사용할 수 있다[24].

4.2.2. Apple M1

Apple M1 프로세서는 최신 ARM 프로세서 중 하나이다. M1은 Mac 및 iPad와 같은 장치에서 사용하기 위해 Apple에서 설계한 프로세서다. M1 프로세서는 CPU, GPU, DSP, 신경망 엔진 등을 하나의 칩에 집적한 일종의 SoC (System on Chip)이다. 5nm 공정으로 생산되며 약 160억 개의 트랜지스터로 구성되어

있다 [13].

ARMv8-A은 고성능 임베디드 애플리케이션용 64-bit 아키텍처이다. 64-bit ARMv8 프로세서는 32-bit (AArch32) 아키텍처와 64-bit (AArch64) 아키텍처 모두를 지원한다. 레지스터 w0-w30의 32-bit 값 또는 x0-x30의 64-bit 값을 유지할 수 있는 31개의 범용 레지스터를 제공한다. ARMv8 프로세서는 2011년 출시되어 스마트폰 시장을 선점하기 시작했으며, 오늘날에는 다양한 스마트폰과 노트북에 널리 사용되고 있다. 프로세서는 주로 임베디드 시스템, 스마트폰, 노트북에 사용되기 때문에 효율적인 구현은 실제 어플리케이션에서 특히 중요하다. ARMv8 프로세서는 강력한 64-bit 효율적인 부호 없는 정수 곱셈 명령을 지원한다. 모듈러 곱셈의 구현은 AArch64 아키텍처를 사용한다[25].

4.2.3. AVX-2

2013년에는 Haswell이라는 최신 인텔 마이크로아키텍처가 출시되었다. 이 아키텍처에는 128-bit 또는 256-bit 레지스터에서 동작하는 AVX2 (Advanced Vector Extensions 2) 명령어 세트가 포함되어 있다. 이전의 AVX와 달리, AVX2는 정수 산술 연산과 레지스터 내의 워드 배열 (8~64비트)을 실행하는 벡터 명령을 가지고 있다. 이러한 명령어를 사용하면 데이터 수준의 병렬화를 이용하여 효율적인 구현을 할 수 있다 [26]. AVX-2는 256-bit 크기의 레지스터를 16개를 지원한다. 이때 각각의 레지스터는 8-bit*32, 16-bit*16, 32-bit*8 벡터로 사용 가능하며, 벡터 명령을 통해 연산이 가능하다.

4.2.4. AVX-512

AVX-512는 AVX (Advanced Vector eXtensions)의 최신 세대이며 32개의 512-bit 레지스터(zmm0 – zmm31)와 다양한 512-bit 명령어로 x64 실행 환경에서 효율적으로 동작한다. AVX-512는 다중 확장으로 구성되며, AVX-512F는 32비트 벡터 곱셈기가 있는 핵심 확장이다. Cannon Lake (Palm Cove 마이크로아키텍처)를 시작으로 Intel은 공개키 암호화 속도를 높이기 위해 특별히 설계된 AVX-512에 소위 Integer Fused Multiply-Add extension (AVX-512IFMA 또는 간단히 IFMA)을 통합했다. 두 개의 새로운 IFMA 명령어 vpmadd52luq 및 vpmadd52huq는 8개의 압축된 부호 없는 52-bit 정수 쌍 (2개의 512비트 벡터의 64비트 요소 각각에 위치)을 곱하여 각각 길이가 104-bit인 8개의 중간 곱을 얻는다. 그런 다음 이러한 결과의 하위 52-bit (vpmadd52luq) 또는 상위 52-bit (vpmadd52huq)가 최종 결과를 보유하는 512-bit 대상 레지스터의 8개의 압축되지 않은 64-bit 정수에 추가된다. AVX-512F의 vpmuludq 및 vpmuldq 곱셈 명령어와 비교하여 IFMA 확장은 52-bit의 더 넓은 곱셈기를 제공할 뿐만 아니라 벡터 곱셈과 벡터 덧셈을 단일 명령어로 결합하였다 [11].

Ⅴ. 양자내성암호 최적구현 사례

[11]은 AVX-512FMA의 대규모 병렬 처리 기능을 사용하여 SIKE 기반 키 캡슐화의 효율성 개선 방법 연구를 진행하였다. 해당 논문에서는 필드 산술, 포인트 산술 및 아이소지니 계산의 병렬화를 목표로 하였다. Operand에 radix-2^{51} 표현을 사용하고 모듈식 축소를 위해 몽고메리의 곱셈 알고리즘을 채택한 $\mathbb{F}_p$, $\mathbb{F}_{p^2}$ 산술 연산을 위해 최적화된 라이브러리를 제시하고 있다. 해당 논문에서는 산술 연산의 두 인스턴스 지연 시간을 최소화 하는 것과 limb-slicing 기법을 사용하여 8개 인스턴스의 처리량을 최대화하는 것을 포함하여 산술 라이브러리의 다양한 변형을 개발하였다. 몽고메리 곡선에서 병렬점 산술 연산을 위한 기술을 설명하고 있으며, 벡터화된 아이소지니 연산 및 키 캡슐화를 위한 구현 기법을 제시하고 있다. AvxSike-LL (낮은 대기시간), AvxSike-HT (높은 처리량) 두 개의 버전을 제공하고 있으며, 두 개의 버전은 비밀에 의존하는 조건문이나 메모리 액세스를 포함하지 않아 타이밍 기반 부채널 공격에 내성이 있다.

SIKEp503 매개변수로 인스턴스화된 LL 버전은 [27]에 제시된 AVX-512IFMA기반 SIKE 소프트웨어 구현보다 약 1.5배 빠른 성능을 보여주고 있으며, Intel Core i3-1005G1 프로세서에서 벤치마킹할 경우 키 생성과 캡슐화 해제 모두에서 Microsoft의 x64 Assembler SIKE 구현보다 약 2.5배 높은 성능을 보여주고 있다. 또한 HT 버전은 Microsoft의 SIKE 라이브러리보다 최대 4.6배 높은 처리량 성능을 보여준다. 자세한 성능 결과는 표 19, 20에서 확인할 수 있다.

〔표 19〕 Intel Core i3-1005G1 프로세서 상의 SIKEp503 구현 결과. op: 연산 종류, K: 키 생성, E: 암호화, D: 복호화.

op	SIDHv3.4 1 instance	Kostic 〔27〕 1 instance	AvxSike-LL 1 instance	AvxSike-HT 8 instances
	Cycles	Cycles	Cycles	Cycles
K	8,078,669	4,842,909	3,215,375	14,179,026
E	13,188,788	7,923,514	4,111,650	22,992,807
D	14,026,750	8,513,409	5,715,005	24,619,263

〔표 20〕 Intel Core i3-1005G1 프로세서 상의 SIKEp434, SIKEp610 구현 결과, op: 연산 종류, K: 키 생성, E: 암호화, D: 복호화.

Scheme	op	SIDHv3.4 1 instance	AvxSike-LL 1 instance	AvxSike-HT 8 instances
		Cycles	Cycles	Cycles
SIKE p434	K	5,976,700	2,474,187	10,442,609
	E	9,690,764	3,062,491	16,801,041
	D	10,357,218	4,341,099	18,053,398
SIKE p610	K	14,096,085	6,918,618	32,172,538
	E	25,875,968	10,001,282	58,747,976
	D	26,040,095	13,124,052	59,103,361
SIKE p751	K	23,843,419	10,212,410	46,662,723
	E	38,446,643	12,804,923	74,885,499
	D	41,368,995	17,834,974	80,684,214

[6]은 Saber를 Cortex-M3와 Cortex-M4 프로세서 상에서 구현하였다. 특히 해당 연구에서는 NTT (Number Theoretic Transform) 연산을 중점적으로 최적화하였다. Saber는 Matrix Vector Multiplication에 대해 NTT를 계산하는 과정에서 많은 시간이 소요된다. 제안하는 기법은 Multi-moduli NTT를 사용하여 연산 시간과 메모리 소요의 트레이드 오프를 가능하게 하였다.

Cortex-M4 상에서는 마스킹 유무에 따라 구현 방법을 두 가지로 나눈다. 마스킹을 사용하지 않을 경우, 16비트에서 32비트로 메모리를 확장하여 구현한다. 마스킹을 사용할 경우에는 big × big 다항식 곱셈을 처리하기 위해서 비대칭 NTT를 사용한다.

Cortex-M3 상에서는 16비트, 32비트 두 가지 방식을 제안한다. 16비트 방식은 [28]의 방식을 하였다. 32비트는 NTT 연산 중 비밀 값이 포함될 때마다 연산 속도와 constant time을 맞추기 위해 public matrix에 non-constant time 연산을 적용하였다.

마지막으로 [28]은 스택 사용을 최적화하지 않았다. [6]에서 제안하는 기법은 스택 사용을 최소로 줄이기 위해 Matrix Vector Multiplication 과정에서 특정 다항식만 메모리상에 유지시킨다. 특정 다항식에서 연산에 필요한 값들을 전부 계산할 수 있으므로, 다른 값들은 메모리에 잔존하지 않고 필요에 따라 연산을 수행한다. 이를 통해 스택 사용량을 최적화하였다. 대신 연산 과정이 추가로 발생함에 따라 연산 시간이 증가하게 된다. 이는 스택을 추가로 사용하는 것으로 연산 시간과 메모리 사이의 트레이드 오프를 지원하게 된다.

표 21, 22는 Cortex-M3, Cortex-M4 상에서 최적 구현물과 비교 알고리즘의 성능 비교를 진행한 것이다.

[8]은 Dilithium, Kyber, Saber 알고리즘을

〔표 21〕 마스크가 없는 Saber의 구현 결과 비교, LS: LightSaber, FS: FireSaber, cc: 클록 사이클(단위: k), st: 스택 크기, K: 키 생성, E: 암호화, D: 복호화.

		LS		Saber		FS	
		cc	st.	cc	st.	cc	st.
pqm3 M3 (speed)	K	710	9652	1328	13252	2171	20116
	E	967	11372	1738	15516	2688	22964
	D	1081	12116	1902	16612	2933	24444
〔6〕 16-bit M3 (speed A)	K	540	5756	939	6788	1439	7812
	E	715	6436	1194	7468	1751	8492
	D	749	6436	1237	7468	1811	8492
〔6〕 16-bit M3 (speed, D)	K	632	3420	1253	3932	1955	4444
	E	887	3204	1614	3332	2427	3460
	D	923	3204	1657	3332	2487	3460
〔6〕 32-bit M3 (speed, A)	K	594	5732	1057	6756	1553	7788
	E	800	6412	1330	7444	1883	8468
	D	877	6420	1429	7452	2016	8476
〔29〕 M4 (stack)	K	612	3564	1230	4348	2046	5116
	E	880	3148	1616	3412	2538	3668
	D	976	3164	1759	3420	2740	3684
〔28〕 M4 (speed)	K	360	14604	658	23284	1008	37116
	E	513	16252	864	32620	1255	40484
	D	498	16996	835	33824	1227	41964
〔6〕 32-bit M4 (speed, A)	K	353	5764	644	6788	990	7812
	E	487	6444	826	7468	1208	8484
	D	456	6440	777	7484	1152	8500
〔6〕 hybrid M4 (stack, D)	K	723	3428	819	3940	1315	4452
	E	597	3204	1063	3332	1617	3468
	D	583	3220	1039	3348	1594	3484

〔표 22〕 마스크가 있는 Saber의 복호화 성능 비교.

	Cycles	Stack
〔30〕	2833k	11656
〔6〕 (speed, A)	2385k	16140
〔6〕 (C)	2615k	10476
〔6〕 (stack, D)	2846k	8432

Cortex-A74과 Apple M1 칩 상에 구현하였다. Cortex-A74와 Apple M1은 ARMv8에 속하는 동일 계열의 프로세서로 제공하는 ISA (Instruction Set Architecture)가 동일하다.

제안하는 기법은 Barret multiplication과 Montgomery multiplication을 NEON SIMD 명령어로 구현하는데 중점을 두었다. 특히 사용하는 명령어의 숫자를 최소화하여 연산에 소요되는 시간을 줄였다. 각 연산에 따라 사용한 명령어의 수는 Barrett multiplication에서 3개, Montgomery multiplication에서 5개 또는 상수가 있을 경우 4개, 반올림을 사용하는 Montgomery multiplication에서 3개, 그리고 숫자가 큰 경우의 Montgomery multiplication에서 7개이다.

또한 Saber, Dilithium, 그리고 Kyber 알고리즘별로 효율적인 NTT 연산을 위해 Butterfly 알고리즘을 NEON SIMD 명령어를 활용하여 구현하였다. 표 23은 구현물의 성능 측정 결과이다.

[21]는 ROLLO-I의 변형인 ROLLO+를 소개하며 AVX2 구현에서의 성능을 제시한다. ROLLO+는 ROLLO-I의 디코딩 알고리즘을 조정하여 일부 유효하지 않은 암호문을 식별하고 거절할 수 있다.

디코딩 알고리즘은 여러 벡터 공간의 교차점을 계산한다. 교집합을 계산하는 한 가지 방법은 Zassennhaus 알고리즘을 사용하는 것이다. 두 벡터 공간의 집합을 생성하는 입력에서 Zassenhaus 알고리즘은 생성 집합에서 행렬을 만들고 가우스를 제거한다. 가우스 제거는 키 생성 및 캡슐화에서 샘플링된 요소의 가중치를 확인하는데 유용하다. 일정한 시간에 가우스 제거를 수행하기 위해 [33]에 제시된 알고리즘을 일반화하여 최초로 도입하였다.

또한 키 생성에서 $\mathbb{F}2mn$에서 한 번의 필드 반전을 포함시켜 2mn-2의 거듭 제곱으로 올리는 것보다 빠르다는 것을 발견하였다. ROLLO+의 레벨 1 매개변수 세트 중 하나인 ROLLO+-I-128 구현은 키 생성에

〔표 23〕 Cortex-A72와 Apple M1 프로세서 상에서의 Kyber, Saber, Dilithium 구현 결과 비교, K: 키 생성, E: 암호화, D: 복호화, (ref): 레퍼런스 Dilithium 코드, (단위: 클록 사이클, k)

Algo.	ref.	CortexA-72			Apple M1		
		K	E	D	K	E	D
kyber 512	〔8〕	62.5	80.7	76.4	14.9	34.8	20.9
	〔31〕	67.9	88.9	87.6	23.0	32.5	29.4
	〔32〕	84.7	109.7	108.6	-	-	-
kyber 768	〔8〕	99.2	127.5	120.7	23.8	36.3	31.0
	〔31〕	110.8	141.3	139.0	36.3	49.2	45.7
	〔32〕	143.8	180.7	179.1	-	-	-
kyber 1024	〔8〕	156.7	192.3	184.2	33.0	48.9	44.0
	〔31〕	176.8	215.7	214.1	55.9	71.6	67.1
	〔32〕	228.1	272.4	270.7	-	-	-
Light saber	〔8〕	64.2	87.3	92.8	20.1	29.7	28.6
	〔31〕	84.0	118.6	136.2	31.2	37.2	35.3
Saber	〔8〕	109.2	140.1	147.9	32.9	44.9	44.1
	〔31〕	175.1	211.4	222.3	50.3	65.4	64.6
Fires aber	〔8〕	175.1	211.4	222.3	50.3	65.4	64.6
	〔31〕	245.2	304.1	330.8	77.0	87.9	86.7
Dilit-hium2	〔8〕	269.7	649.2	272.8	71.1	224.1	69.8
	(ref)	410.3	1354	449.6	187.8	741.1	199.6
Dilit-hium3	〔8〕	515.8	1089	447.5	152.4	365.2	104.8
	(ref)	743.2	2309	728.9	358.8	1218	329.2
Dilit-hium5	〔8〕	782.8	1437	764.9	178.1	426.6	167.5
	(ref)	1152	2904	1199	544.8	1531	557.7

851,823 Cycle, 캡슐화에 30,361 Cycle, 캡슐화 해제에 673,666 Cycle를 사용하였다. [34]의 최신 ROLLO-I-128 구현과 비교하여 Encapsulation의 경우 10.6배, Decapsulation의 경우 14.5배 빨라졌다. [35]의 BIKE 레벨 1 매개변수 세트의 최신 구현과 비교할 때 키 생성 시간은 1.4배 느리지만 캡슐화 시간은 3.8배 빠르며 캡슐화 해제 시간은 2.4배 빠르다. 자세한 성능은 표 24에서 확인할 수 있다.

[14]는 UoV와 Rainbow를 AVX2 명령어 셋을 사용하여 구현하였다. 주된 구현 사항으로는 이차 다항식 연산을 사용하는 vinegar 값을 선형 연산으로 바꾸는 것이다. 또한 vinegar 값을 central polynomial 값으로 치환하여 사용한다.

선형 연산을 빠르게 계산하기 위해서 Block Matrix Inversion을 사용한다. 기본적으로 UoV와 Rainbow는

가우시안 소거법을 사용하는데, 이때 사용되는 행렬의 크기가 매우 크기 때문에 연산에 소요되는 시간 또한 증가하게 된다. 본 연구에서는 Block Matrix Inversion은 역행렬의 크기를 줄이는 것으로 연산에 소요되는 시간을 줄인다. 추가로 서명할 메시지가 주어지기 전에 일부 연산을 미리 계산해두는 사전 연산이 가능하다. 이는 일부 연산이 메시지와는 독립적으로 연산이 가능한 UoV에 적용이 가능하다. Rainbow의 경우에도 UoV에 적용한 기법과 동일하게 적용이 가능하나, 첫 번째 레이어에서만 적용이 가능하다. 표 25는 최적구현된 UoV와 Rainbow의 성능을 측정한 것이다.

〔표 24〕 Intel Core i7-8850H(Coffee Lake) 상에서의 ROLLO-I〔36〕, Intel Xeon E3-1220v5(Skylake) 상에서의 ROLLO^{+}, BIKE〔35〕의 키 생성, 암호화, 복호화 과정에 대한 사이클 수 비교(ROLLO-II은 구현하지 않음, 단위: 클록 사이클).

Instance	key gen.	encap.	decap	level
R I-128	11,034,623	984432	9775241	1
	11,204,649	320835	9744693	
R^{+} I-128	851,823	30,361	673,666	1
R^{+} I-192	980,860	38,748	878,398	3
R^{+} I-256	1,477,519	55,353	1,635,966	5
R^{+} II-128	4,663,096	70,621	876,533	1
R^{+} II-192	4,058,419	94,138	1,060,271	3
R^{+} II-256	4,947,630	90,021	1,497,315	5
bike L1	589,625	114,256	1,643,551	1
bike L3	1668,511	267,644	5,128,078	3

〔표 25〕 UoV와 Rainbow의 성능 비교, Lv: 보안 강도 성능평가 단위: 클록 사이클, k, 메모리 단위: bytes.

Scheme	Category	Lv.I	Lv.III	Lv.V
UoV	Sign w/o precomp.	201.8	708.0	1486
	Precomp. (offline)	189.2	690.6	1460
	Precomp (online)	11.8	19.4	23.1
	Total	201.0	710.0	1483
	Memory cost	2256	5402	9504
Rainbow	Sign w/o precomp.	68.2	322.8	807.3
	Precomp. (offline)	37.2	173.2	508.9
	Precomp (online)	32.0	142.2	278.5
	Total	69.2	315.4	787.4
	Memory cost	2152	4792	5648

Ⅵ. 결 론

본 고에서는 NIST에서 진행하고 있는 양자내성암호 표준화 공모전의 다양한 양자내성암호에 대해서 확인해 보았다. 이와 더불어 해당 양자내성암호의 복잡한 연산을, 자원이 한정된 사물인터넷 환경 상에서 구현한 방법 및 구현 결과물의 성능을 확인해 보았다. 해당 구현 기법에 대한 최신 동향은 현재 국내에서 진행되고 있는 양자내성암호 표준화 공모전에 적용하여 성능을 개선하는데 활용 가능할 것으로 사료된다. 본 고에서 살펴본 바와 같이 빠르게 발전해 가고 있는 양자컴퓨터와 양자내성암호에 보다 관심을 가지고 지속적인 연구가 되어야 할 것으로 사료된다.

참 고 문 헌

[1] R.P.Feynman, “Simulating physics with computers,” *International Journal of Theoretical Physics*, 21, pp.467–488 Jun 1982.

[2] F.Arute, K.Arya, R.Babbush, D.Bacon, J.C.Bardin, R.Barends, and J.M.Martinis, “Quantum supremacy using a programmable superconducting processor,” *Nature*, 574(7779), pp.505-510, Oct 2019.

[3] J.Chow, O.Dial and J.Gambetta, “IBM Quantum breaks the 100 qubit processor barrier,” *IBM Research Blog*, Nov 2021.

[4] S.Jaques, M.Naehrig, M.Roetteler, and F.Virdia, “Implementing Grover oracles for quantum key search on AES and LowMC,” *In Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, Cham, pp.280-310, May 2020.

[5] P.W.Shor, “Algorithms for quantum computation: discrete logarithms and factoring,” In Proceedings 35th annual symposium on foundations of computer science, Ieee, pp.124-134, Nov 1994.

[6] A.Abdulrahman, J.P.Chen, Y.J.Chen, V.Hwang, M.J. Kannwischer, and B.Y.Yang, “Multi-moduli NTTs for saber on Cortex-M3 and Cortex-M4,” *Cryptology ePrint Archive*, Jul 2021.

[7] 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.” *International Conference on Cryptology in Africa*, pp.282-305, 2018.

[8] H.Becker, V.Hwang, M.J.Kannwischer, B.Y.Yang, and S.Y.Yang, “Neon NTT: Faster Dilithium, Kyber, and Saber on Cortex-A72 and Apple M1,” *Cryptology ePrint Archive*, Nov 2021.

[9] D.Jao, and L.D.Feo, “Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies,” *In International Workshop on Post-Quantum Cryptography, Springer, Berlin*, Heidelberg, pp. 19-34, Nov 2011.

[10] D. Jao, et al, “Supersingular Isogeny Key Encapsulation,” NIST PQC Round 3 submission, Oct. 1, 2020.

[11] H.Cheng, G.Fotiadis, J.Großschädl, and P.Y.Ryan, “Highly vectorized SIKE for AVX-512,” *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp.41-68, Feb 2022.

[12] J.Ding, B.Y.Yang, C.H.O.Chen, M.S.Chen, and C.M.Cheng, “New differential-algebraic attacks and reparametrization of rainbow,” *International Conference on Applied Cryptography and Network Security*. Springer, Berlin, Heidelberg, pp.242-257, 2008.

[13] H.Kwon, H.Kim, M.Sim, W.K.Lee, and H.Seo, “Look-up the Rainbow: Efficient Table-based Parallel Implementation of Rainbow Signature on 64-bit ARMv8 Processors,” *Cryptology ePrint Archive*, Jul 2021

[14] K.A.Shim, S.Lee, and N.Koo, “Efficient Implementations of Rainbow and UOV using AVX2,” *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp.245-269. Nov 2022.

[15] J.Patarin, “The oil and vinegar signature scheme,” *In Dagstuhl Workshop on Cryptography September*, Sep 1997.

[16] A.Kipnis, and A.Shamir, “Cryptanalysis of the oil and vinegar signature scheme,” *In Annual international cryptology conference, Springer*, Berlin, Heidelberg, pp.257-266, Aug 1998.

[17] A.Kipnis, J.Patarin, and L.Goubin, “Unbalanced oil and vinegar signature schemes,” In International Conference on the Theory and Applications of *Cryptographic Techniques, Springer, Berlin*, Heidelberg, pp.206-222, May 1999.

[18] N.Courtois, L.Goubin, W.Meier, and J.D.Tacier, “Solving underdefined systems of multivariate quadratic equations,” *In International Workshop on Public Key Cryptography*, Springer, Berlin, Heidelberg. pp.211-227, Feb 2002.

[19] C.A.Melchor, N.Aragon, M.Bardet, S.Bettaieb, L.Bidoux, O.Blazy, J.C.Deneuville, P.Gaborit, A.Hauteville, A.Otmani, O.Ruatta, J.P.Tillich, and G.Zemor, “ROLLO – Rank-Ouroboros, LAKE& LOCKER,” *Submission to the NIST Post Quantum Standardization Process, Round 2*, Apr 2019.

[20] E.Gorla, “Rank-metric codes”, *Journal of Algebraic Combinatorics*, 52(1), pp.1-19, 2020.

[21] T.Chou, and J.H. Liou, “A Constant-time AVX2 Implementation of a Variant of ROLLO,” IACR *Transactions on Cryptographic Hardware and Embedded Systems*, vol 2022, no. 1, pp. 152-174. Nov 2021.

[22] S.Y.Lee, G.S.Yoo, “Implementation of IoT Sensor Communication Platformusing ARM Cortex—M4,” *The Journal of Korean Association of Computer Education*, 25(1), pp.283-285, 2021.

[23] H.J.Seo, “High Speed Implementation of LEA on

ARM Cortex-M3 processor," *Journal of the Korea Institute of Information and Communication Engineering*, 22(8), pp.1133-1138, 2018.

[24] I.Lin, B.Jeff, and I.Rickard, "ARM platform for performance and power efficiency－Hardware and software perspectives," *2016 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, pp.1-5, 2016.

[25] H.J.Seo, P.Sanal, and W.K.Lee, "No Silver Bullet: Optimized Montgomery Multiplication on Various 64-Bit ARM Platforms," *International Conference on Information Security Applications*. Springer, Cham, pp.194-205, 2021.

[26] C.Roberto, and J.López. "Software implementation of SHA-3 family using AVX2," *Simpósio Brasileiro em Segurança da Informaçao e de Sistemas Computacionais 14 (2014)*, pp.330-333, 2014.

[27] D.Kostic and S.Gueron. "Using the new VPMADD instructions for the new post quantum key encapsulation mechanism SIKE," *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*, pp.215-218, 2019.

[28] C.M.M.Chung, V.Hwang, M.J.Kannwischer, G.Seiler, C.J.Shih, and B.Y.Yang. "NTT multiplication for NTT- unfriendly rings new speed records for saber and NTRU on Cortex-M4 and AVX2," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp.159-188, Feb 2021.

[29] J.M.Bermudo Mera, A.Karmakar, and I.Verbauwhede. "Time-memory trade-off in Toom－Cook multiplication: an application to module-lattice based cryptography," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp.222-244, Feb 2020.

[30] M.V.Beirendonck, J.P.D'Anvers, A.Karmakar, J.Balasch, and I.Verbauwhede. "A side-channel resistant implementation of SABER," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 17(2) pp.1-26, 2021.

[31] D.T.Nguyen and K.Gaj. "Optimized software implementations of CRYSTALS-Kyber, NTRU, and Saber using NEON-based special instructions of ARMv8," Proceedings of the NIST 3rd PQC Standardization Conference (NIST PQC 2021). 2021.

[32] P.Sanal, E.Karagoz, H.Seo, R.Azarderakhsh, and M.M.Kermani. "Kyber on ARM64: compact implementations of Kyber on 64-bit ARM Cortex-A processors," *Cryptology ePrint Archive*, Report 2021/561, 2021.

[33] D.J.Bernstein, T.Chou, and Peter Schwabe. "Mcbits: fast constant- time code-based cryptography," *Cryptographic Hardware and Embedded Systems - CHES 2013,* pp.250-272, Springer, 2013.

[34] C.Aguilar-Melchor, N.Aragon, S.Bettaieb, L.Bidoux, O.Blazy, J.C.Deneuville, P.Gaborit, G.Zémor, A.Couvreur, and A.Hauteville. "RQC," 2020. Available: https://pqc-rqc.org/.

[35] M.S.Chen, T.Chou, and M.Krausz. "Optimizing BIKE for the Intel Haswell and ARM Cortex-M4", *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp.97-124, 2021.

[36] C.Aguilar-Melchor, N.Aragon, E.Bellini, F.Caullery, R.H.Makarim, and C.Marcolla. "Constant time algorithms for ROLLO-I-128", *SN Computer Science*, 2(5), pp.1-19, 2021.

〈저 자 소 개〉

권 혁 동 (HyeokDong Kwon)

정회원

2018년 2월 : 한성대학교 정보시스템공학과 졸업

2020년 2월 : 한성대학교 IT융합공학부 석사

2020년 3월~현재 : 한성대학교 정보컴퓨터공학과 박사과정

<관심분야> 정보보호, 암호구현

엄 시 우 (Siwoo Eum)

정회원

2021년 2월 : 한성대학교 IT융합공학부 학사 졸업

2021년 3월~현재 : 한성대학교 IT융합공학부 석사과정

<관심분야> 암호구현, 정보보호, 인공지능

심 민 주 (MinJoo Sim)

정회원

2021년 2월 : 한성대학교 IT융합공학부 학사 졸업

2021년 3월~현재 : 한성대학교 IT융합공학부 석사과정

<관심분야> 암호구현, 정보보호

서 화 정 (Hwajeong Seo)

종신회원

2010년 2월 : 부산대학교 컴퓨터공학과 졸업

2012년 2월 : 부산대학교 컴퓨터공학과 석사

2015년 4월~5월 : 싱가포르 난양공대 인턴쉽

2016년 2월 : 부산대학교 컴퓨터공학과 박사

2017년 3월 : 싱가포르 과학기술청 연구원

2017년 4월 ~ 현재 : 한성대학교 조교수

<관심분야> 암호구현