Code based PQC

서화정, 장경배, 최승주

HANSUNG UNIVERSITY 한성대학교

CryptoCraft LAB
https://crypto.modoo.at

Contents

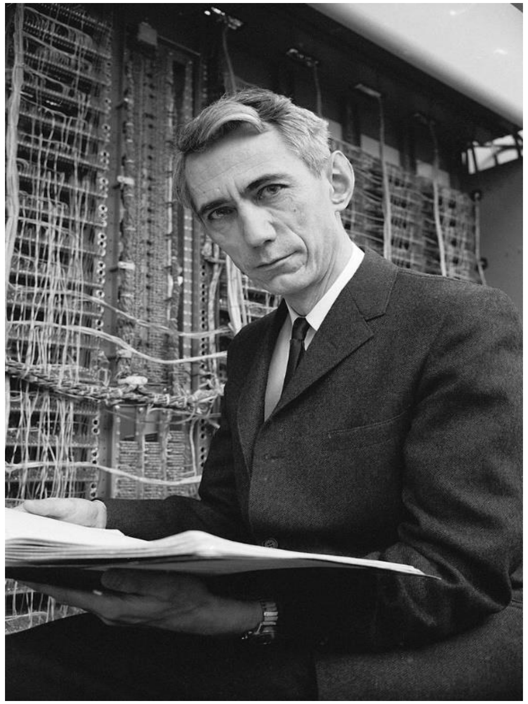
CryptoCraft LAB

Contents

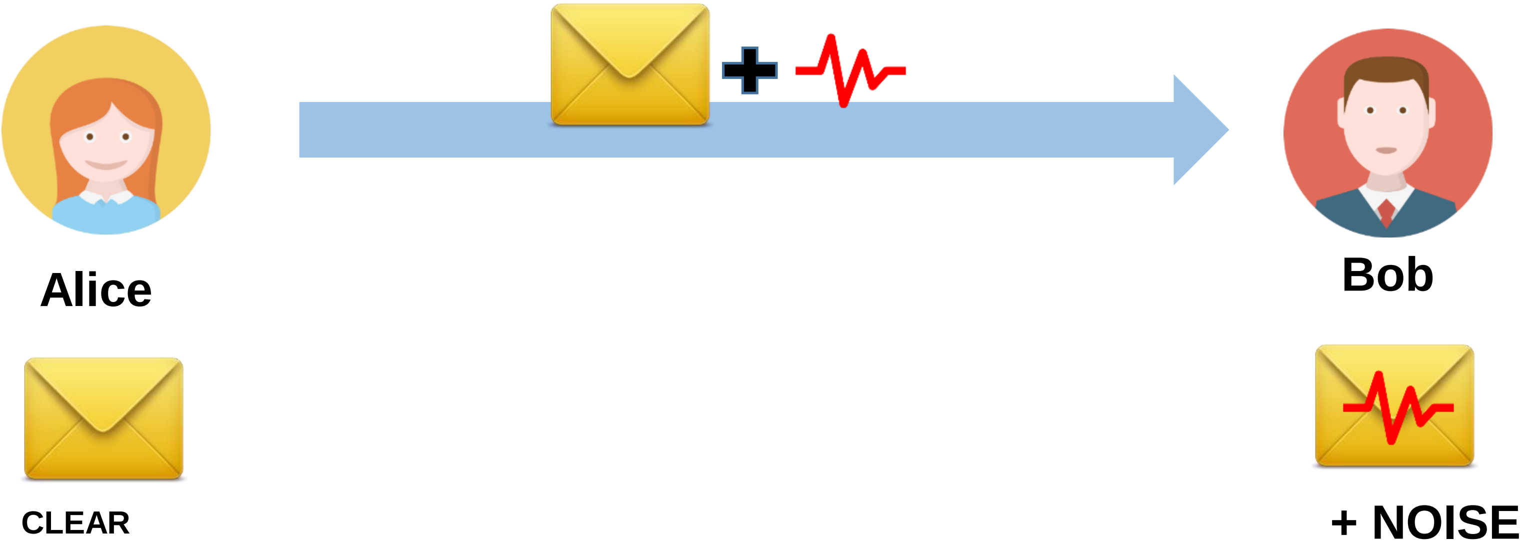
CryptoCraft LAB

코드 이론

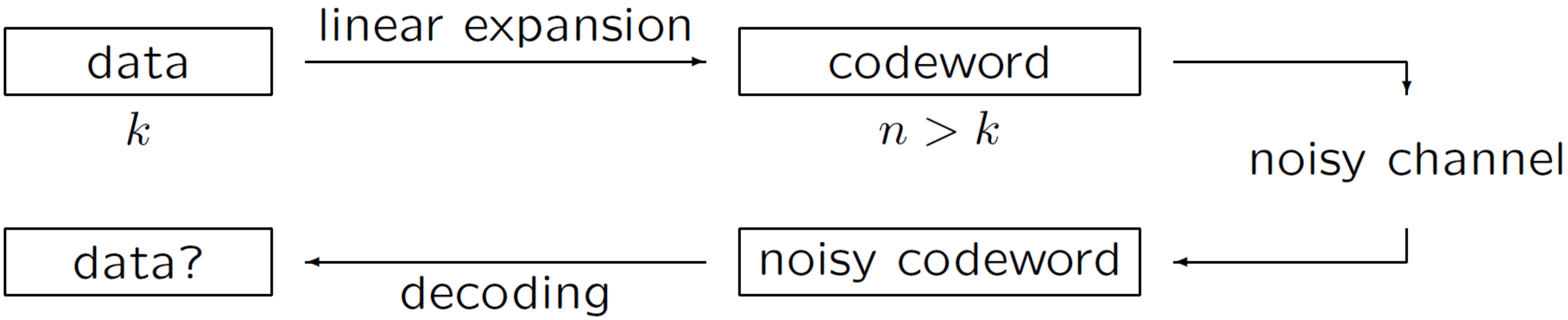
- **코드 이론**
  - 불완전한 채널에서 생기는 잡음을 수정하기 위해 제안
  - 오류 수정 능력을 가진 코드를 사용하여 메시지 전달



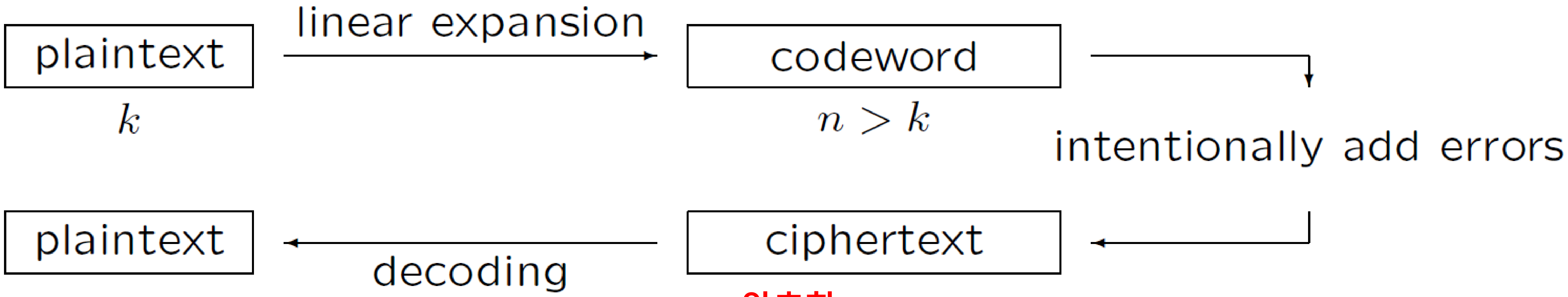
Claude Shannon



통신과 암호 상의 코드



통신



암호화

안전한 Code Family

Family	제안	공격
Goppa	**McEliece (78)**	**-**
Reed-Solomon	Niederreiter (86)	Sidelnikov & Chestakov (92)
Concatenated	Niederreiter (86)	Sendrier (98)
Reed-Muller	Sidelnikov (94)	Minder & Shokrollahi (07)
AG codes	Janwa & Moreno (96)	Faure & Minder (08) Couvreur, Marquez-Corbella & Pellikaan (14)
LDPC	Monico, Rosenthal & Shokrollahi (00)	
Convolutional codes	Londahl & Johansson (12)	Landais & Tillich (13)

[Faugere, Gauthier, Otmani, Perret & Tillich 11] binary Goppa code 공격

PQC Round2

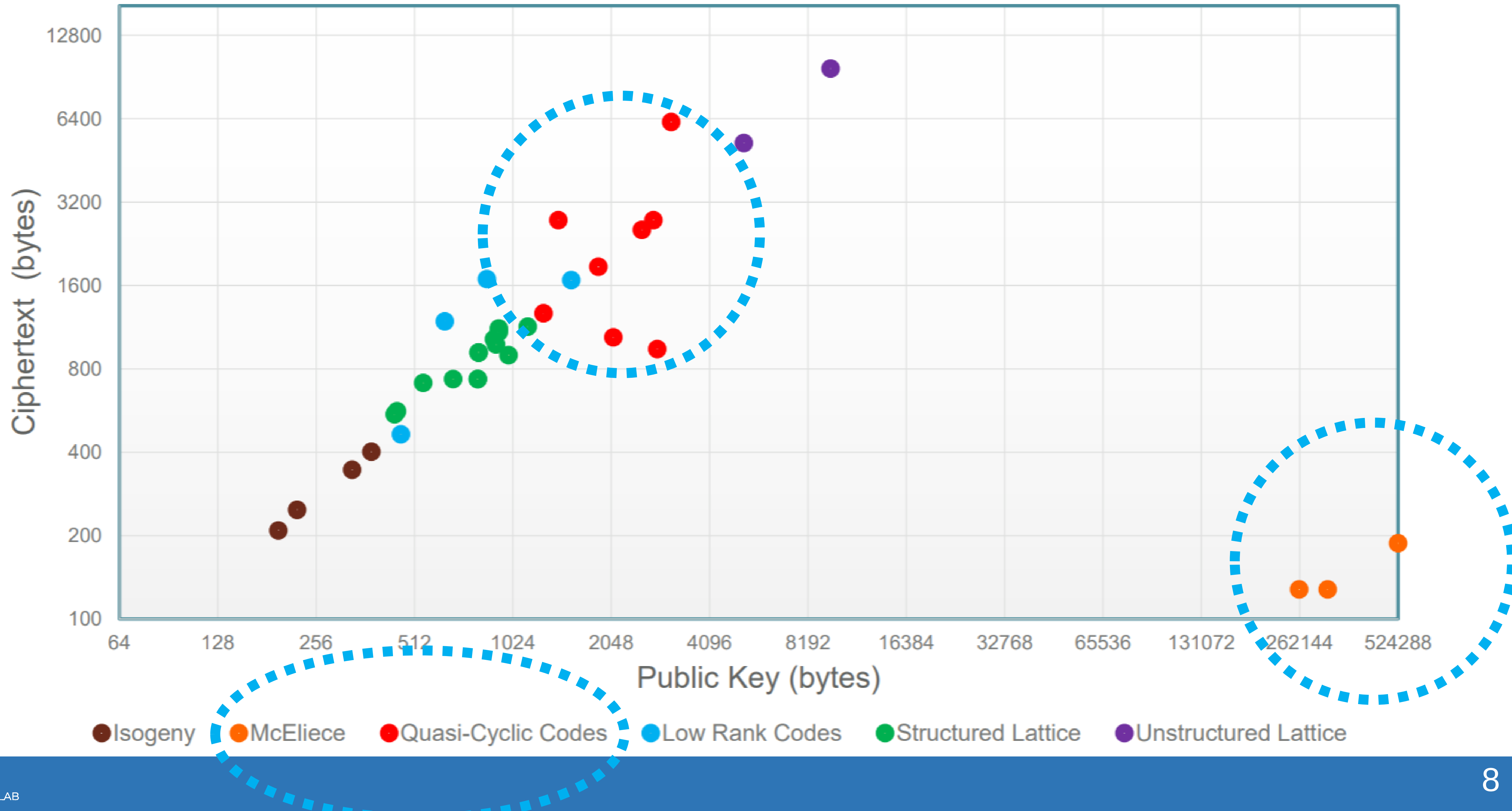
NIST PQC 라운드#1

알고리즘	기반 코드
Classic McEliece	Goppa
Big Quake	Goppa
NTS-KEM	Goppa
BIKE	Short Hamming
HQC	Short Hamming
QC-MDPC KEM	Short Hamming
LEDApkc	Short Hamming
LEDAkem	Short Hamming
LOCKER	Low Rank
LAKE	Low Rank
Ouroboros-R	Low Rank
RQC	Low Rank

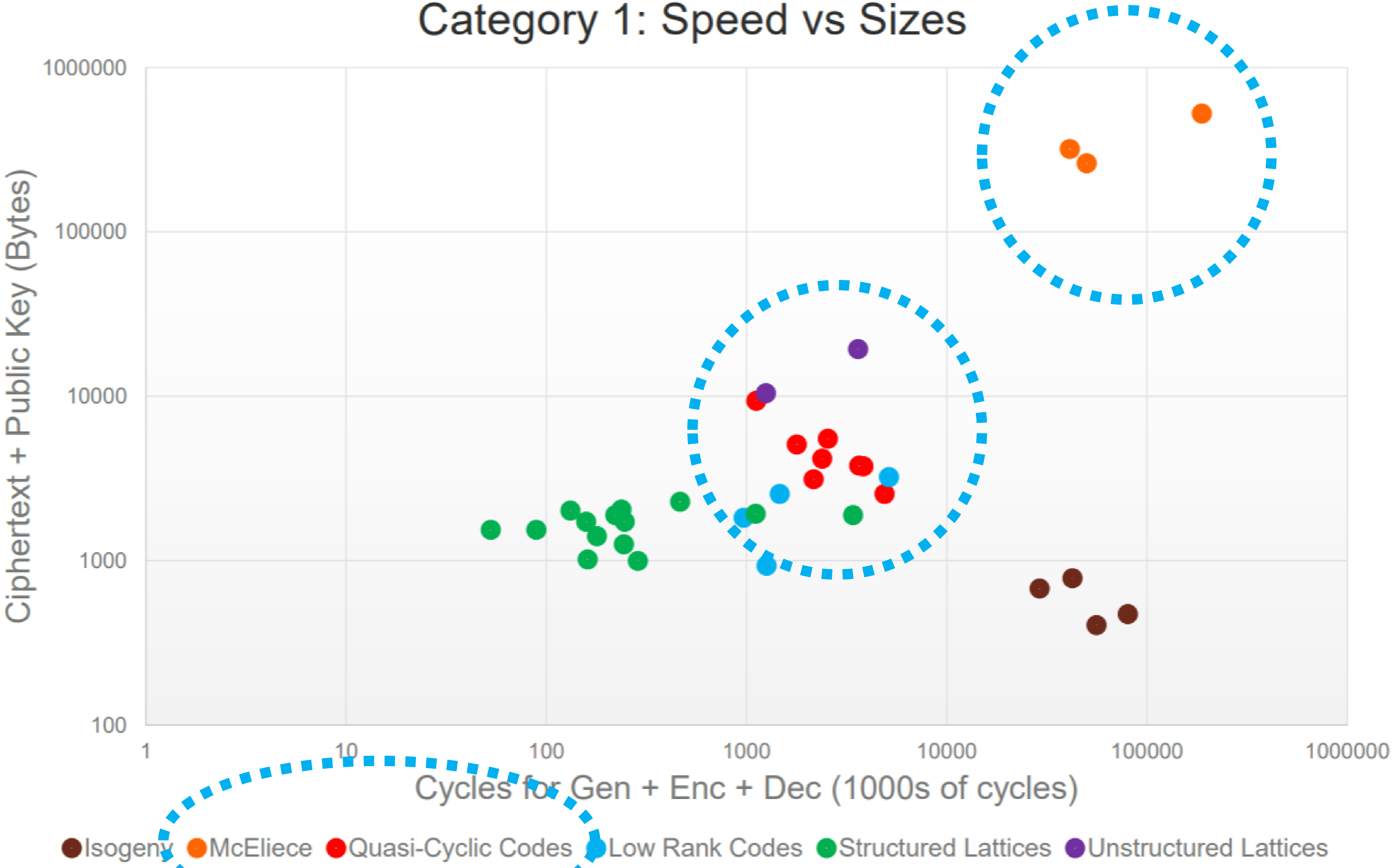
NIST PQC 라운드#2

알고리즘	기반 코드
Classic McEliece	**Goppa**
NTS-KEM	Goppa
BIKE	**Short Hamming**
HQC	Short Hamming
LEDAcrypt	**Short Hamming**
Rollo	Low Rank
RQC	Low Rank

Category 1: Public Key vs Ciphertext size - PKE/KEMs



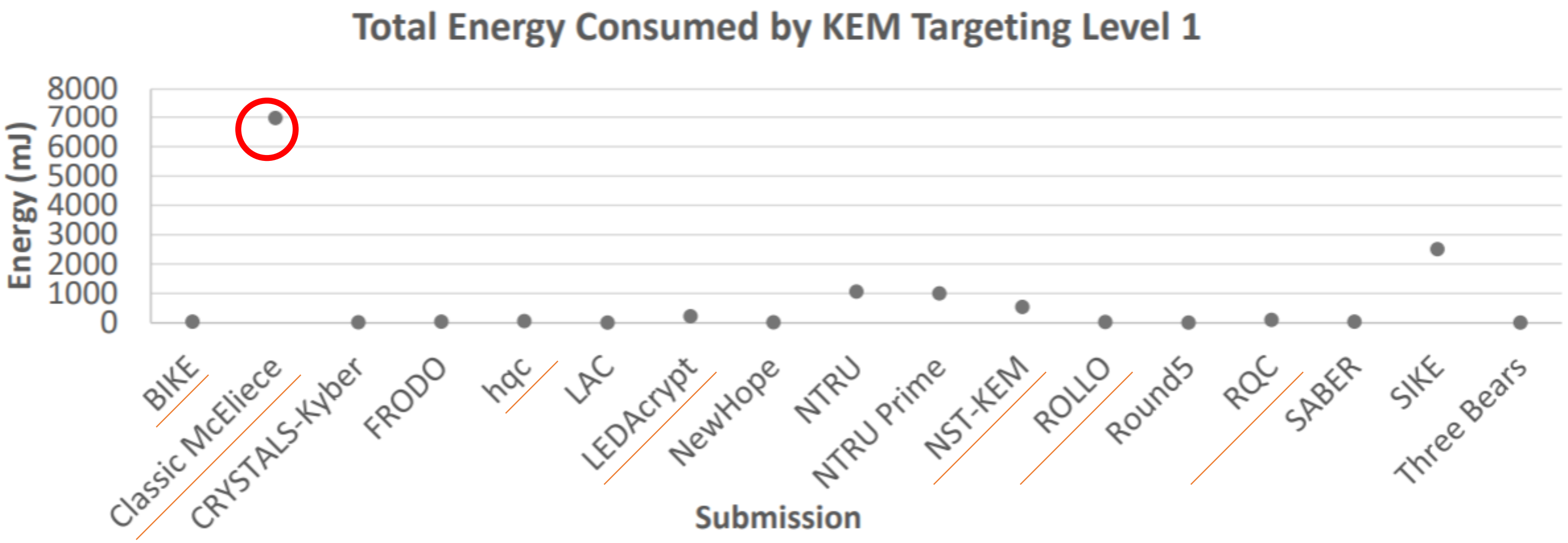
Category 1: Speed vs Sizes



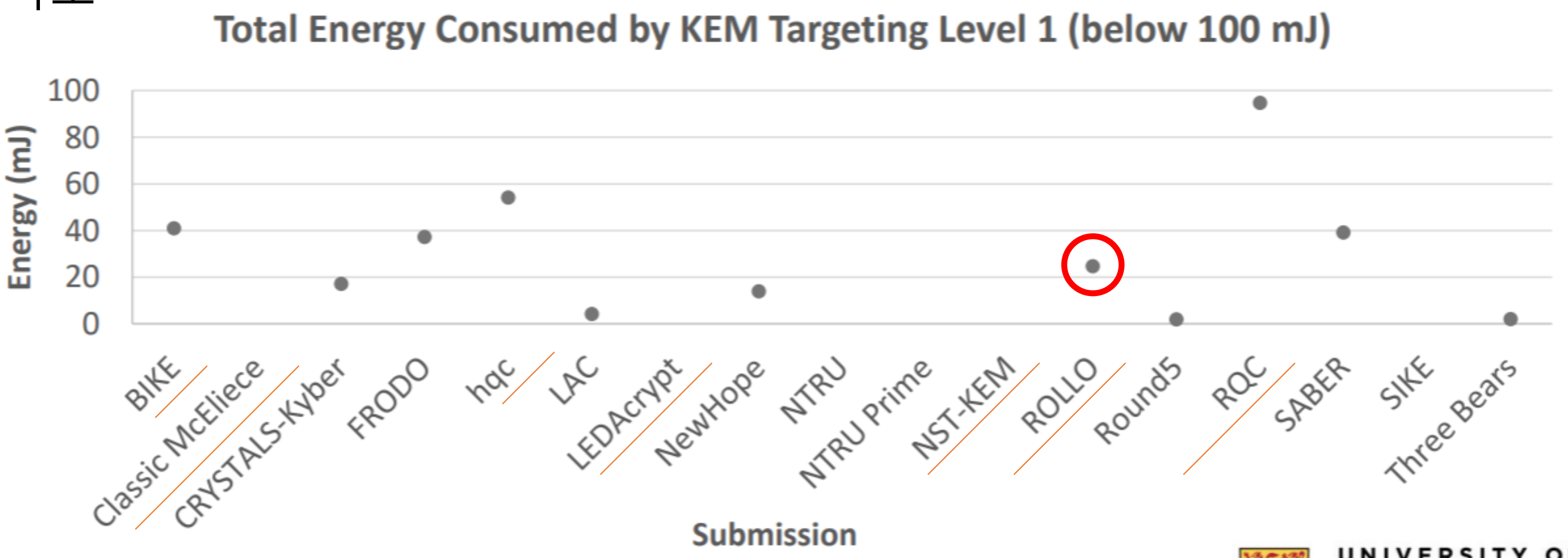
KEM/PKE 구현 현황

- 기본적인 레퍼런스 구현 이외의 최적화 코드 보유 현황

Scheme	x86 Assembly Optimization			Other Hardware		
	SIMD	AES-NI	Other	ARM	FPGA	ASIC
BIKE	O	O	O		O	
Classic McEliece	O					
HQC	O					
LEDAcrypt	O					
NTS-KEM	O		O			
ROLLO						
RQC						



에너지 소비량 == 연산속도



UNIVERSITY OF WATERLOO

12

Contents

CryptoCraft LAB

Classic McEliece

1. 1981 Clark–Cain [18], crediting Omura.
2. 1988 Lee–Brickell [33].
3. 1988 Leon [34].
4. 1989 Krouk [32].
5. 1989 Stern [52].
6. 1989 Dumer [24].
7. 1990 Coffey–Goodman [19].
8. 1990 van Tilburg [55].
9. 1991 Dumer [25].
10. 1991 Coffey–Goodman–Farrell [20].
11. 1993 Chabanne–Courteau [15].
12. 1993 Chabaud [16].
13. 1994 van Tilburg [56].
14. 1994 Canteaut–Chabanne [11].
15. 1998 Canteaut–Chabaud [12].
16. 1998 Canteaut–Sendrier [13].
17. 2008 Bernstein–Lange–Peters [8].
18. 2009 Bernstein–Lange–Peters–van Tilborg [10].
19. 2009 Finiasz–Sendrier [27].
20. 2011 Bernstein–Lange–Peters [9].
21. 2011 May–Meurer–Thomae [37].
22. 2012 Becker–Joux–May–Meurer [3].
23. 2013 Hamdaoui–Sendrier [29].
24. 2015 May–Ozerov [38].
25. 2016 Canto Torres–Sendrier [54].

- **최초의 코드기반 암호 McEliece 와 Niederreiter 의 듀얼버전**
- **KEM(Key Encapsulation Mechanism) 으로 설계 되었음**
- **1978년 이후, 코드기반암호를 연구한 점점 더 정교한 공격이 발표되었음**

  Effect → 동일한 키 사이즈로 동일한 보안성을 달성

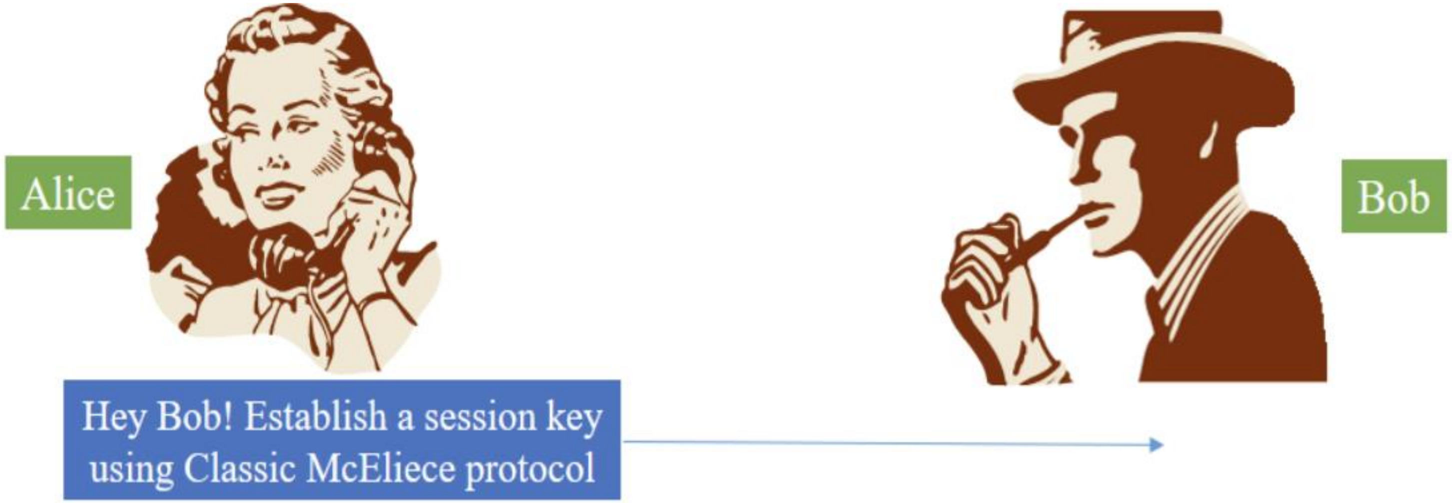
- **Classic McEliece 팀의 제출에 대한 주요쟁점은 보안**

  40년동안 안전성을 지켜온 McEliece의 Goppa code 를 사용

  자신들의 보안성을 훼손시키지 않는 선에서 효율성을 향상

Classic McEliece – Key Generation (1 / 10)

- Alice 는 Bob에게 Classic McEliece Protocol 을 사용하여 session key 성립을 요청



extension field $GF(2^m)$ / $k(x)$

- 이 때 extension field $GF(2^m)$ 와 유한체의 원소를 결정할 root polynomial $k(x)$ 는 공개정보

Classic McEliece – Key Generation (2 / 10)

- $GF(2^4)$ / $k(x)$

2^4 개의 유한개의 원소를 찾기위해 $X^{15} = 1$ 을 만족하는 irreducible polynomial 을 찾아야 함

$$X^{15}-1 = (X+1)(X^2+X+1)(X^4+X+1)(X^4+X^3+1)(X^4+X^3+X^2+X+1).$$

ex) root polynomial : $k(x)$ → (X^4+X^3+1)

$$\mathbb{F}_{2^4} = \frac{\mathbb{F}_2[x]}{\langle x^4 + x^3 + 1 \rangle} = \boxed{\mathbb{F}_2(\beta)}$$

Classic McEliece – Key Generation (3 / 10)

$k(x)$ → (X^4+X^3+1)

이제 $\beta^4 = \beta^3 + 1$ 을 사용하여 다음 $GF(2^4)^*$ 를 찾아낼 수 있음

Classic McEliece – Key Generation (4 / 10)

$$
\begin{array}{lcccccccccl}
1 & = & 1 & & & & & & & = & (1,0,0,0)^T; \\
\beta & = & & & \beta & & & & & = & (0,1,0,0)^T; \\
\beta^2 & = & & & & & \beta^2 & & & = & (0,0,1,0)^T; \\
\beta^3 & = & & & & & & & \beta^3 & = & (0,0,0,1)^T; \\
\beta^4 & = & 1 & & & & & + & \beta^3 & = & (1,0,0,1)^T; \\
\beta^5 & = & 1 & + & \beta & & & + & \beta^3 & = & (1,1,0,1)^T; \\
\beta^6 & = & 1 & + & \beta & + & \beta^2 & + & \beta^3 & = & (1,1,1,1)^T; \\
\beta^7 & = & 1 & + & \beta & + & \beta^2 & & & = & (1,1,1,0)^T; \\
\beta^8 & = & & & \beta & + & \beta^2 & + & \beta^3 & = & (0,1,1,1)^T; \\
\beta^9 & = & 1 & & & + & \beta^2 & & & = & (1,0,1,0)^T; \\
\beta^{10} & = & & & \beta & & & + & \beta^3 & = & (0,1,0,1)^T; \\
\beta^{11} & = & 1 & & & + & \beta^2 & + & \beta^3 & = & (1,0,1,1)^T; \\
\beta^{12} & = & 1 & + & \beta & & & & & = & (1,1,0,0)^T; \\
\beta^{13} & = & & & \beta & + & \beta^2 & & & = & (0,1,1,0)^T; \\
\beta^{14} & = & & & & & \beta^2 & + & \beta^3 & = & (0,0,1,1)^T.
\end{array}
$$

$^*\beta^4 = \beta^3 + 1$

$$
\begin{aligned}
\beta^5 &= \beta^4 \cdot \beta \\
&= (\beta^3 + 1) \cdot \beta \\
&= \beta^4 + \beta \\
&= 1 + \beta + \beta^3
\end{aligned}
$$

- 위와 같이 순환 구조의 유한체 원소 형성

Classic McEliece – Key Generation (5 / 10)

$\mathbb{F}_2(\beta) = \{0, 1, \beta, \beta^2, \ldots, \beta^{14}\}$

- $\text{Goppa code } \Gamma(L, g(z))$ 를 정의할 수 있음

1. Bob은 개인키로 monic & irreducible 한 t 차 다항식 $g(z)$ 를 생성, 이 때 t 는 최대로 수정할 수 있는 오류의 개수

   $g(z) = z^2 + z + \beta$ → 최대 2개의 오류 수정 가능

   * message x G = codeword
   parity-check H

2. 위와 같은 유한체 $\mathbb{F}_2(\beta)$의 원소에서 랜덤하게 n개의 원소를 순서대로 선택하여 subset $L = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$을 구성

   $L = \mathbb{F}_2(\beta) = \{0, 1, \beta, \beta^2, \ldots, \beta^{14}\}$ (n = 16)

Classic McEliece – Key Generation (6 / 10)

3. $t \times n$ (2×16)의 $parity - check$ 행렬 $H = \{h_{i,j}\}$ 를 계산,

$$\begin{matrix} (g_2\alpha_i + g_1)/g(\alpha_j) \\ (g_2)/g(\alpha_j) \end{matrix} \Rightarrow H = \begin{pmatrix} \boxed{(0+1)/g(\alpha_1)} & (1+1)/g(\alpha_2) & \cdots & (1+1)/g(\alpha_{16}) \\ 1/g(\alpha_1) & 1/g(\alpha_2) & \cdots & 1/g(\alpha_{16}) \end{pmatrix}$$

$$\boxed{h_{1,1}} = g(0)^{-1} = (\beta)^{-1} = \beta^{14}$$

* $g(z) = z^2 + z + \beta$

$g_1 = 1, \ g_2 = 1$

subset $L = \{\alpha_1, \alpha_2, \dots, \alpha_n\} = \{0, 1, \beta, \beta^2, \dots, \beta^{14}\}$

$$H = \begin{pmatrix} \beta^{14} & 0 & \beta^{10} & \beta^{3} & \beta^{10} & \beta^{9} & \beta^{13} & 1 & \beta^{9} & \beta^{13} & \beta^{11} & \beta^{8} & \beta^{11} & \beta^{14} & \beta^{3} & \beta^{8} \\ \beta^{14} & \beta^{14} & \beta^{13} & \beta^{9} & \beta^{6} & \beta^{6} & \beta^{3} & \beta^{7} & \beta^{11} & \beta^{7} & \beta^{9} & \beta^{3} & \beta^{12} & \beta^{13} & \beta^{11} & \beta^{12} \end{pmatrix}$$

Classic McEliece – Key Generation (7 / 10)

4. 앞서 생성한 parity-check 행렬 H 를 각 원소에 해당하는 bit 로 변환

$$\begin{pmatrix} \boxed{\begin{matrix}\beta^{14} \\ \beta^{14}\end{matrix}} & \begin{matrix}0 \\ \beta^{14}\end{matrix} & \begin{matrix}\beta^{10} \\ \beta^{13}\end{matrix} & \begin{matrix}\beta^{3} \\ \beta^{9}\end{matrix} & \begin{matrix}\beta^{10} \\ \beta^{6}\end{matrix} & \begin{matrix}\beta^{9} \\ \beta^{6}\end{matrix} & \begin{matrix}\beta^{13} \\ \beta^{3}\end{matrix} & \begin{matrix}1 \\ \beta^{7}\end{matrix} & \begin{matrix}\beta^{9} \\ \beta^{11}\end{matrix} & \begin{matrix}\beta^{13} \\ \beta^{7}\end{matrix} & \begin{matrix}\beta^{11} \\ \beta^{9}\end{matrix} & \begin{matrix}\beta^{8} \\ \beta^{3}\end{matrix} & \begin{matrix}\beta^{11} \\ \beta^{12}\end{matrix} & \begin{matrix}\beta^{14} \\ \beta^{13}\end{matrix} & \begin{matrix}\beta^{3} \\ \beta^{11}\end{matrix} & \begin{matrix}\beta^{8} \\ \beta^{12}\end{matrix} \end{pmatrix}$$

$$\Downarrow$$

$$H = \begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\
1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0
\end{pmatrix}$$

Classic McEliece – Key Generation (8 / 10)

$$
\begin{array}{lcccccccccl}
1 & = & 1 & & & & & & & = & (1,0,0,0)^T; \\
\beta & = & & & \beta & & & & & = & (0,1,0,0)^T; \\
\beta^2 & = & & & & & \beta^2 & & & = & (0,0,1,0)^T; \\
\beta^3 & = & & & & & & & \beta^3 & = & (0,0,0,1)^T; \\
\beta^4 & = & 1 & & & & & + & \beta^3 & = & (1,0,0,1)^T; \\
\beta^5 & = & 1 & + & \beta & & & + & \beta^3 & = & (1,1,0,1)^T; \\
\beta^6 & = & 1 & + & \beta & + & \beta^2 & + & \beta^3 & = & (1,1,1,1)^T; \\
\beta^7 & = & 1 & + & \beta & + & \beta^2 & & & = & (1,1,1,0)^T; \\
\beta^8 & = & & & \beta & + & \beta^2 & + & \beta^3 & = & (0,1,1,1)^T; \\
\beta^9 & = & 1 & & & + & \beta^2 & & & = & (1,0,1,0)^T; \\
\beta^{10} & = & & & \beta & & & + & \beta^3 & = & (0,1,0,1)^T; \\
\beta^{11} & = & 1 & & & + & \beta^2 & + & \beta^3 & = & (1,0,1,1)^T; \\
\beta^{12} & = & 1 & + & \beta & & & & & = & (1,1,0,0)^T; \\
\beta^{13} & = & & & \beta & + & \beta^2 & & & = & (0,1,1,0)^T; \\
\boxed{\beta^{14}} & = & & & & & \beta^2 & + & \beta^3 & = & (0,0,1,1)^T.
\end{array}
$$

Classic McEliece – Key Generation (9 / 10)

5. 앞서 생성한 parity-check 행렬 H 를 가우스 소거(Gaussian elimination) 를 수행하여 아래과 같이 systematic form 으로 변환 → 후에 Decapsulation 시 사용됨

$$H$$

$$\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\
1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0
\end{pmatrix}
\Rightarrow
(\boxed{I_{n-k}} \mid \boxed{T})$$

$$\left(\begin{array}{cccccccc|cccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0
\end{array}\right)$$

Identity-matrix
$(n-k \times n-k)$

Classic McEliece – Key Generation (10 / 10)

6. T 를 공개키로 사용, Bob은 랜덤하게 n-bit 벡터 s 생성, s = (0000000000000000)
↳ 후에 Encapsulation 시 사용

7. $\Gamma = (g, \alpha_1, \alpha_2, \ldots, \alpha_n)$ 그리고 Bob 의 개인키는 (s, Γ)

- Private key

s = (0000000000000000)

$g(z) = z^2 + z + \beta$

$L = \mathbb{F}_2(\beta) = \{0, 1, \beta, \beta^2, \ldots, \beta^{14}\}$

- Public key

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Classic McEliece – Encoding (1 / 1)

- Alice는 인코딩 과정에서 두가지 입력 값이 필요 : weight – t 인 n-bit 벡터 e 그리고 공개키 T

1. $H = (I_{n-k} \mid T)$ 을 사용하여 인코딩
2. $C_0 = He^T$
3. return C_0 (n – k) – bit

Challenge

$C_0 = He^T$ 라는 신드롬 계산 식에서 C_0 와 H 가 주어진다 해도

low – weight 벡터 e 를 찾아내기 매우 어려움

CryptoCraft LAB

Classic McEliece – Decoding (1 / 1)

$C_0 = He^T$ 를 이용하여

Bob 은 수신한 C_0 를 디코딩하여 Hamming weight $\mathrm{wt}(e) = t$ 인 벡터 e 를 복구해야 함

Decoding Subroutine

1. $(n - k)$ - bit 벡터 C_0 에 k – bit 만큼 zero 를 패딩하여 아래와 같은 n – bit 의 벡터 v 로 확장

$$C_0 \text{ to } v = (C_0, \underbrace{0, \ldots, 0}_{k\text{ – bit}})$$

2. Bob 은 자신의 개인키 $\Gamma = (g, \alpha_1, \alpha_2, \ldots, \alpha_n)$ 를 가지고 $\mathrm{wt}(e) = t$ 의 벡터 e 를 찾아낼 수 있음

Summary – Key Generation, Encoding, Decoding

Alice

Hey Bob! Establish a session key using Classic McEliece protocol

Bob

Generates Classic McEliece Key Pair

- Private key : $\Gamma = \{s, g(z), a_1, a_2, \ldots, a_n\}$
- Public key : T

Public key : T

Random Plaintext : e

Using $H = (I_{n-k} \mid T)$,

Compute $C_0 = He^T$

Classic McEliece – Encapsulation (1 / 2)

1. Alice는 weight – t 인 n-bit 벡터 e 를 생성 ➡ $\mathbf{e} = (1100000000000000)$, length n = 16, weight t = 2

2. Public key T 를 사용하여 C_0 를 계산 ➡ $C_0 = H\mathbf{e} = (11000000)$

3. $C_1 = \mathrm{H}(2, e)$ 를 계산, 그리고 Chipertext $C = (C_0, C_1)$ 구성

   이 때 사용하는 해시함수는 SHA-256, 해시 입력 값 2 는 Byte로 표현 (i.e. 00000010)

$C_1 = \mathrm{H}(2, \mathbf{e}) = 26fe36f811ac8fe9f19ba997a39d3682ef06b29509cca1903ffe4a0b247c833f$
00100110111111100011011011111000000100011010110010001111111010011111000110011011101
0100110101011110100011100111010011011010000010111011110011010110010100101010100111001
1001010000110010001111111111110010010100101100100100011111001000001100111111.

$C = (C_0, C_1) =$
11000000001001101111111000110110111110000001000110101100100011111110100111110001100
1101110101001100101111010001110011101001101101000001011101111001101011001010010101010
1001110011001010000110010001111111111110010010100101100100100011111001000001100111
111.

Classic McEliece – Encapsulation (2 / 2)

4. $K = \mathsf{H}(1, e, C)$ 를 계산

$$K = \mathtt{H}(1, \mathbf{e}, C) = 90\text{d}7\text{c}9\text{dccc}4689\text{f}6894\text{b}1\text{b}6\text{e}58\text{ee}9\text{b}3832\ 8\text{e}4\text{df}9937536\text{eb}9\text{b}5715\text{a}38\text{ee}4\text{e}1\text{be}$$

5. Alice는 Session key K 출력, 그리고 Ciphertext C 를 Bob 에게 전송

Classic McEliece – Decapsulation (1 / 2)

- Bob 은 수신한 Ciphertext C 로부터 Session Key K 를 decapsulate 해야함

1. Ciphertext C 를 (C_0, C_1) 로 나눈다.

$C = (C_0, C_1) =$
$\boxed{11000000}$001001101111111000110110111110000001000110101100100011111110100111110001100
1101110101001100101111010001110011101001101101000000101110111100110101100101001010
1001110011001010000110010001111111111110010010100101100100100011111001000001100111
111.

$$C_0 = H\mathbf{e} = (11000000)$$

2. Set $b \leftarrow 1$

$^*C_0 = H\mathbf{e}$

3. C_0 에 대해 private key $\Gamma = \{s, \boxed{g(z), a_1, a_2, \ldots, a_n}\}$ 를 사용한 디코딩 알고리즘으로 plaintext e 를 복구

   만약 디코딩 알고리즘이 $\perp$ 를 return 한다면, set $e \leftarrow s$, $b \leftarrow 0$

Classic McEliece – Decoding Algorithm (1 / 5)

- $C_0 = H\mathbf{e} = (11000000)$

1. $\mathbf{v} = (C_0, 00000000) = (1100000000000000)$ 와 같이 k – bit 의 zero 벡터로 패딩

2. 벡터 $\mathbf{V}$ 에서 t 개의 오류를 수정

   해당 Goppa code 의 원본 codeword c 가 있었고,

   c 의 두 자리 bit 에 오류가 생긴 벡터가 $\mathbf{V}$ 라 가정하고, 오류수정을 진행

Classic McEliece – Decoding Algorithm (2 / 5)

3. Goppa code 를 사용한 오류수정의 핵심 방정식

$S(\mathbf{z})\sigma(z) \equiv w(z) \mod g(z)$ 을 풂으로써 오류 수정

키 생성시 Goppa Code 로 생성한 H 를 사용하여 벡터 $\mathbf{V}$ 의 Syndrome 값 계산

$$\mathbf{v} = (C_0, 00000000) = (\boxed{11}00000000000000)$$

$$H = \begin{pmatrix} \boxed{\begin{matrix} \beta^{14} & 0 \\ \beta^{14} & \beta^{14} \end{matrix}} & \begin{matrix} \beta^{10} & \beta^{3} & \beta^{10} & \beta^{9} & \beta^{13} & 1 & \beta^{9} & \beta^{13} & \beta^{11} & \beta^{8} & \beta^{11} & \beta^{14} & \beta^{3} & \beta^{8} \\ \beta^{13} & \beta^{9} & \beta^{6} & \beta^{6} & \beta^{3} & \beta^{7} & \beta^{11} & \beta^{7} & \beta^{9} & \beta^{3} & \beta^{12} & \beta^{13} & \beta^{11} & \beta^{12} \end{matrix} \end{pmatrix}$$

$S(\mathbf{v}) = \beta^{14}$ → Syndrome 값

$\sigma(z) = (z + \sigma_1)(z + \sigma_2)$ → 오류 위치 다항식

$w(z) = 1$ → 오류 값 (Binary 이므로 1)

Classic McEliece – Decoding Algorithm (3 / 5)

- 즉, 다음 방정식을 풂으로 써 오류 위치를 찾을 수 있음

$$\beta^{14}(z+\sigma_1)(z+\sigma_2) \equiv 1 \; mod \; g(z)$$

$$\beta^{14}z^2 + (\sigma_1+\sigma_2)z\beta^{14} + \sigma_1\sigma_2 + \underline{\beta^{14}(z^2+z+\beta)} \equiv 1 \; mod \; g(z)$$

* $g(z) = z^2 + z + \beta$

$$(\sigma_1+\sigma_2+1)z\beta^{14} + \sigma_1\sigma_2\,\beta^{14} + \underline{\beta^{15}} \equiv 1 \; mod \; g(z)$$

$$\sigma_1+\sigma_2 = 1, \quad \sigma_1\sigma_2 = 0$$

$$\therefore \; \sigma_1 = 0, \; \sigma_2 = 1$$

오류 위치에 따라 수신한 벡터 $\mathbf{v}$ 의 오류를 수정하여 Goppa code 의 원본 codeword 를 복구할 수 있음

$\mathbf{v}$ = (1100000000000000) → $\mathbf{c}$ = (0000000000000000)

Classic McEliece – Decoding Algorithm (4 / 5)

- Question. 다음과 같은 오류 수정을 하는 이유?

Bob은 수신한 C_0 로부터 plaintext $\mathbf{e}$ 를 복구해야 함

$$C_0 = H\mathbf{e}$$

디코딩 알고리즘을 모르는 수신자는 low-weight codeword $\mathbf{e}$ 를 찾아내기 매우 어려움

→ Finding low-weight codeword problem

하지만 디코딩 알고리즘을 사용하면 아래 식으로 간단하게 복구 가능

$$\mathbf{e} = \mathbf{v} + \mathbf{c} = (1100000000000000)$$

Classic McEliece – Decoding Algorithm (5 / 5)

$\mathbf{e} = v + c = (1100000000000000)$ $\quad C_0 = H\mathbf{e} = (11000000)$

$$He = H(v+c) = Hv + \boxed{Hc} = Hv = C_0$$

↳ 원본 코드워드의 syndrome 값은 0

$Hv = C_0$?

∵ 앞서 $H = (I_{n-k} \mid T)$ 그리고 $\mathbf{v} = (C_0, 00000000) = (1100000000000000)$

마지막으로 c 는 v 에 대한 오류수정으로 t 개의 bit 가 수정 되었기 때문에 $v + c$ 의 weight 는 t

Classic McEliece – Decapsulation (2 / 2)

- Alice의 Encapsulation 과정과 동일하게 Sesseion key K를 Bob 또한 획득하여 키 교환이 완료

4. $C_1' = \mathsf{H}(2, e)$ 를 계산, 만약 $C_1' \neq C_1$ 라면, $\text{set } e \leftarrow s$, $b \leftarrow 0$
5. $K = \mathsf{H}(b, e, C)$ 를 계산
6. Session key K 출력

구현의 관점에서 보았을 때, 부채널 공격으로 인한 비밀정보의 노출을 피해야 함

연산과정에서 Success 와 failure 구분의 특징이 드러나지 않기 위해

$\perp$ 를 반환해도 연산을 멈추지 않고 계속 진행하기를 언급

Classic McEliece – Conclusion

Alice

Bob

Hey Bob! Establish a session key using Classic McEliece protocol

Generates Classic McEliece Key Pair

- Private key : $\Gamma = \{s, g(z), a_1, a_2, \ldots, a_n\}$
- Public key : T

Public key : T

Random Plaintext : e
Using $H = (I_{n-k} \mid T)$
Compute $C_0 = He^T$
Session Key : $K = H(1, e, C)$

Ciphertext: C

Decrypt C to obtain e
Session Key : $K = H(1, e, C)$

Contents

CryptoCraft LAB

BIKE(Bit Flipping Key Encapsulation)

- **QC-MDPC(Quasi-Cyclic Moderate Density Parity-Check)** 부호에 기반한 암호화 알고리즘
- **Bit Flipping Decoding** 방식을 이용해 복호화 진행

BIKE - 표기

NOTATION	DESCRIPTION
$\mathbb{F}_2$:	Finite field of 2 elements.
$\mathcal{R}$:	The cyclic polynomial ring $\mathbb{F}_2[X]/\langle X^r - 1\rangle$.
$\lvert v \rvert$:	The Hamming weight of a binary polynomial v.
$u \xleftarrow{\$} U$:	Variable u is sampled uniformly at random from set U.
h_j:	The j-th column of a matrix H, as a row vector.
$\star$:	The component-wise product of vectors.

Table 1: Notation

BIKE - 정의

- 선형 부호: 이진 선형부호 *C(nxk)*

  길이:*n*, 차원:*k*

- 생성자와 패리티 검사 행렬

  - 행렬 $G \in \mathbb{F}_2^{k \times n}$ 는 이진선형부호 *C(nxk)*로 부터 나온 생성자 행렬
  - 행렬 $H \in \mathbb{F}_2^{(n-k) \times n}$ 는 *C*의 패리티 $c = mG$
  - 벡터 *m*과 *G*를 갖고 코드워크 생성:

- 벡터 *e*에 대한 신드롬 값: $s^T = He^T$

BIKE – Quasi-Cyclic Codes

- **순환행렬**

  - 행 벡터가 선행 행 벡터에 비례하여 오른쪽으로 하나만큼 이동한 행렬

  - 첫번째 행에 의해 전체 행렬이 정의됨

- **블록순환행렬**

  - 동일한 크기의 순환 행렬로 구성

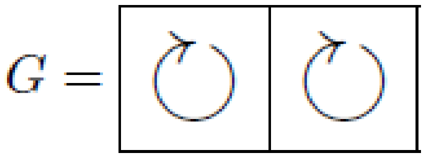
  - 크기: order(주기)

  - 한 행에 들어있는 순환행렬의 개수: index

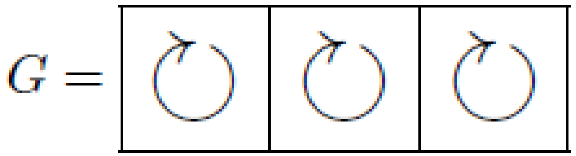
BIKE – Quasi-Cyclic Codes

- 준순환부호

  - index n_0와 order r인 이진 순환부호는 index n_0 및 order r의 블록 순환 행렬을 생성기 행렬로서 허용하는 선형 부호
  - (n_0, k_0)QC 부호는 index n_0, 길이 n_0r 및 k_0r 차원으로 구성된 순환부호



The rows of G span a $(2, 1)$-QC code



The rows of G span a $(3, 1)$-QC code

BIKE – QC-MDPC

- 이진 MDPC(Moderate Density Parity Check)

  - 주기O($\sqrt{n}$)의 밀도를 갖는 페리티 검사 행렬을 사용하는 이진 선형 코드

  - 원격 통신에서 오류 정정을 위해 사용되는 LDPC(Low Density Parity Check)와 사용되는 것과 유사한 반복적인 디코더를 사용

  - t = O($\sqrt{n}$ log n)만큼의 에러 수정 가능

BIKE – QC-MDPC

- (n_0, k_0) quasi-cyclic code
- 길이 $n = n_0 r$
- 차원 $k = k_0 r$
- 주기 r (index n_0)
- 무게 $w = O(\sqrt{n})$

  패리티 체크 행렬의 행 무게

BIKE – Message Protocol

Alice **Bob**

BIKE – Message Protocol

Alice **Bob**

1. 임시적으로 사용하는 QC-MDPC 키 쌍(sk, pk) 생성
   - 개인키: sk, 공개키: pk

BIKE – Message Protocol

Alice **Bob**

1. 임시적으로 사용하는 QC-MDPC 키 쌍(sk, pk) 생성
   - 개인키: sk, 공개키: pk

2. 전송(pk) →

3. 에러 백터 e 생성
4. 에러 백터 e로부터 세션키(대칭) K 추출
5. pk를 사용해 e 암호화 → 암호문 ct 생성

BIKE – Message Protocol

Alice **Bob**

1. 임시적으로 사용하는 QC-MDPC 키 쌍(sk, pk) 생성
   - 개인키: sk, 공개키: pk

2. 전송(pk) →

3. 에러 백터 e 생성
4. 에러 백터 e로부터 세션키(대칭) K 추출
5. pk를 사용해 e 암호화 → 암호문 ct 생성

6. 전송(ct) ←

7. sk를 사용해 ct를 복호화해서 e나 ⊥(실패 신호) 추출
8. 에러 백터 e로부터 세션키(대칭) K 추출

BIKE-1,2,3

- IND-CPA 보안성을 보장하는 3가지 BIKE 버전 존재

  - BIKE-1, BIKE-2, BIKE-3

- 메시지 교환시 일어나는 키 교환에서 임시 키 사용

  - Forward Secuiry 성취

  - 디코딩 실패 관찰을 이용한 공격에 대한 대비

*선택평문공격에 대한 비구별성(Indistinguishability under chosen plaintext attack; IND-CPA)

BIKE 1

- McEliece의 변형을 사용함으로서 빠르게 키 생성이 가능
- QC-MDPC McEliece와는 다르게 개인키인 순환 블록의 Inverse를 계산하지 않고 전체 개인 행렬에 곱하여 체계적인 형태를 얻어내는 연산을 하지 않음
  - 랜덤한 순환 블록을 개인 순환 행렬에 곱해 개인 코드 구조를 숨김
- 코드(code word)에 메시지를 포함하지 않고 오류벡터에 메시지를 포함하여 전송

BIKE-1 KeyGen

- Input: λ , taget quantum security level
- Output: private key(h_0, h_1) and public key(f_0, f_1)

BIKE-1 KeyGen

- Input: λ , taget quantum security level
- Output: private key(h_0, h_1) and public key(f_0, f_1)

0. λ 가 주어지면 r, w 설정

   r: order, w: weight

BIKE-1 KeyGen

- Input: λ , taget quantum security level
- Output: private key(h_0, h_1) and public key(f_0, f_1)

0. λ 가 주어지면 r, w 설정

1. 개인키 h_0, h_1 생성

- h_0, h_1 무게 = w/2 → 홀수

- h_0과 h_1은 R로 부터 랜덤하게 선출

* $\mathcal{R}$: The cyclic polynomial ring $\mathbb{F}_2[X]/\langle X^r - 1\rangle$

BIKE-1 KeyGen

- Input: λ , taget quantum security level
- Output: private key(h_0, h_1) and public key(f_0, f_1)

0. λ 가 주어지면 r, w 설정

1. 개인키 h_0, h_1 생성

2. g 생성

- g는 R로 부터 랜덤하게 선출
- 무게는 홀수(r/2)

BIKE-1 KeyGen

- Input: λ , taget quantum security level
- Output: private key(h_0, h_1) and public key(f_0, f_1)

0. λ 가 주어지면 r, w 설정
1. 개인키 h_0, h_1 생성
2. g 생성
3. gh_1, gh_0 → f_0, f_1

BIKE-1 Encaps

- Input: public key f_0, f_1
- Output: the encapsulated key *K* and the cryptogram c

BIKE-1 Encaps

- Input: public key f_0, f_1
- Output: the encapsulated key *K* and the cryptogram c

1. R^2 공간에서 e_0과 e_1 벡터 선택 ($e_0 + e_1 = t$)

BIKE-1 Encaps

- Input: public key f_0, f_1
- Output: the encapsulated key *K* and the cryptogram c

1. R^2 공간에서 e_0과 e_1 벡터 선택 ($e_0 + e_1 = t$)
2. R에서 랜덤하게 벡터 m 생성

BIKE-1 Encaps

- Input: public key f_0, f_1
- Output: the encapsulated key *K* and the cryptogram c

1. R^2 공간에서 e_0과 e_1 벡터 선택 ($e_0 + e_1 = t$)
2. R에서 랜덤하게 벡터 m 생성
3. $c = (c_0, c_1) \leftarrow (mf_0 + e_0, mf_1 + e_1)$ 연산하여 암호문 생성

BIKE-1 Encaps

- Input: public key f_0, f_1
- Output: the encapsulated key *K* and the cryptogram c

1. R^2 공간에서 e_0과 e_1 벡터 선택 ($e_0 + e_1 = t$)
2. R에서 랜덤하게 벡터 m 생성
3. $c = (c_0, c_1) \leftarrow (mf_0 + e_0, mf_1 + e_1)$ 연산하여 암호문 생성
4. $K \leftarrow \mathbf{K}(e_0, e_1)$ e_0, e_1으로 세션키 생성

   *__K__: SHA256 해시 함수

BIKE-1 Decaps

- Input: private key h_0, h_1 and cryptogram c
- Output: decapsulated key K or failure symbol $\perp$

BIKE-1 Decaps

- Input: private key h_0, h_1 and cryptogram c
- Output: decapsulated key *K* or failure symbol ⊥

1. c를 c0과 c1으로 나누고 신드롬 값 연산 $s \leftarrow c_0h_0 + c_1h_1$

BIKE-1 Decaps

- Input: private key h_0, h_1 and cryptogram c
- Output: decapsulated key *K* or failure symbol ⊥

1. c를 c0과 c1으로 나누고 신드롬 값 연산 s ← $c_0h_0 + c_1h_1$
2. 에러 백터 e_0', e_1' 을 추출하기 위해 s를 decode(Bit Flipping Decoding)

BIKE-1 Decaps

- Input: private key h_0, h_1 and cryptogram c
- Output: decapsulated key *K* or failure symbol ⊥

1. c를 c0과 c1으로 나누고 신드롬 값 연산 s ← $c_0h_0 + c_1h_1$
2. 에러 벡터 e_0', e_1' 을 추출하기 위해 s를 decode(Bit Flipping Decoding)
3. 만약 decode 해서 나온 (e_0', e_1')가 t가 안되거나 decoding이 실패하면 실패 신호(⊥) 반환 후 정지

*Encap: 1. R^2 공간에서 e_0과 e_1 벡터 선택 ($e_0 + e_1 = t$)

BIKE-1 Decaps

- Input: private key h_0, h_1 and cryptogram c
- Output: decapsulated key *K* or failure symbol ⊥

1. c를 c0과 c1으로 나누고 신드롬 값 연산 s ← $c_0h_0 + c_1h_1$
2. 에러 백터 e_0’, e_1’ 을 추출하기 위해 s를 decode(Bit Flipping Decoding)
3. 만약 decode 해서 나온 (e_0’, e_1’)가 t가 안되거나 decoding이 실패하면 실패 신호(⊥**)** 반환 후 정지
4. Decode 성공했다면 나온 e_0’와 e_1’을 갖고 *K* ← **K**(e_0’, e_1’) 연산 해서 *K* 획득

BIKE-1 Decaps

- Bit Flipping Decoding

BIKE-1 Decaps

- Bit Flipping Decoding

  example

전송 메시지

X = 0 0 0 0 0 0 0

도착 메시지

Y = 0 1 0 0 1 0 0

BIKE-1 Decaps

- Bit Flipping Decoding

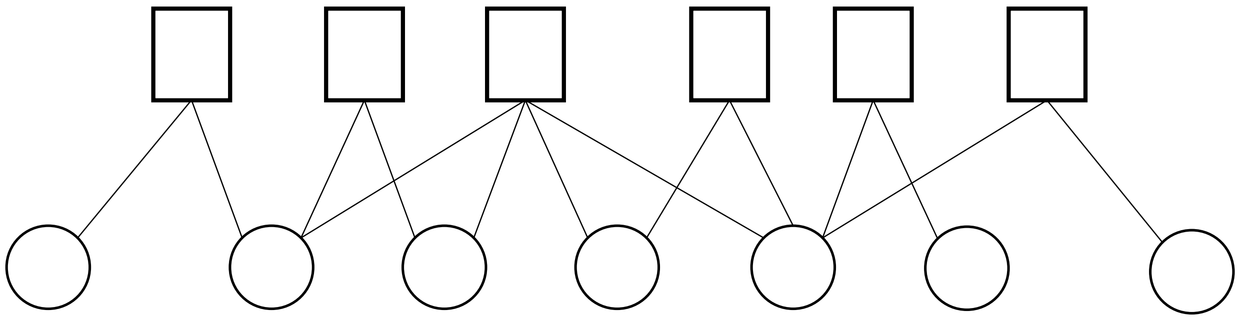
example

전송 메시지 도착 메시지

X = 0 0 0 0 0 0 0 Y = 0 1 0 0 1 0 0

$$H = \begin{matrix} 1&1&0&0&0&0&0 \\ 0&1&1&0&0&0&0 \\ 0&1&1&1&1&0&0 \\ 0&0&0&1&1&0&0 \\ 0&0&0&0&1&1&0 \\ 0&0&0&0&1&0&1 \end{matrix}$$

Tanner graph



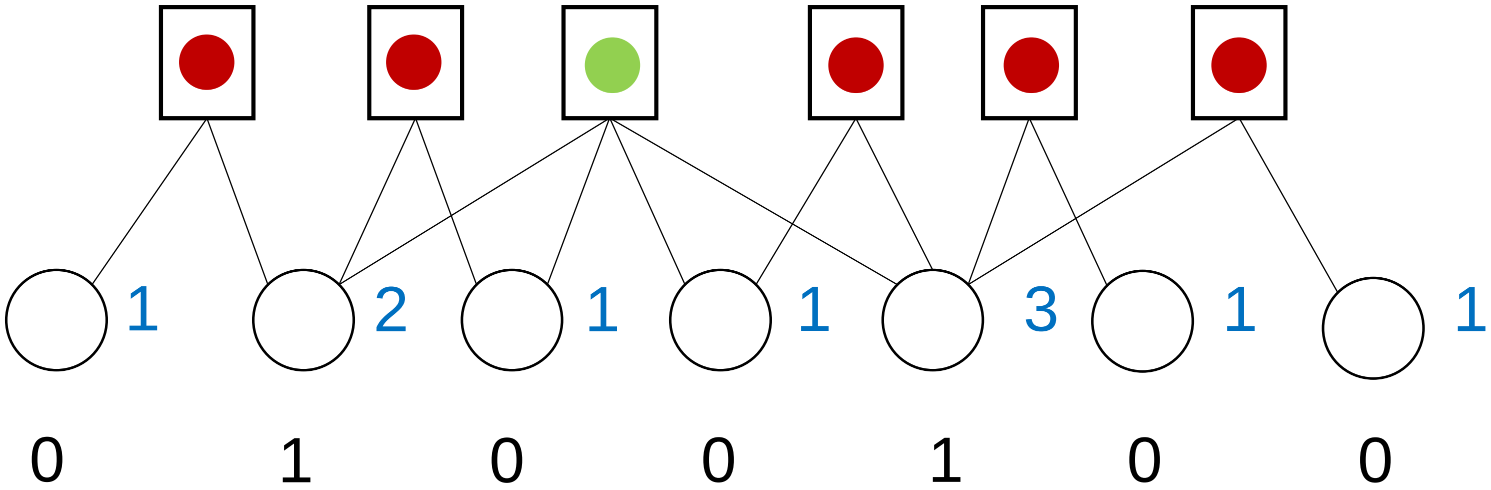
BIKE-1 Decaps

- Bit Flipping Decoding

  example

전송 메시지　　　　도착 메시지

X = 0 0 0 0 0 0 0　Y = 0 1 0 0 1 0 0



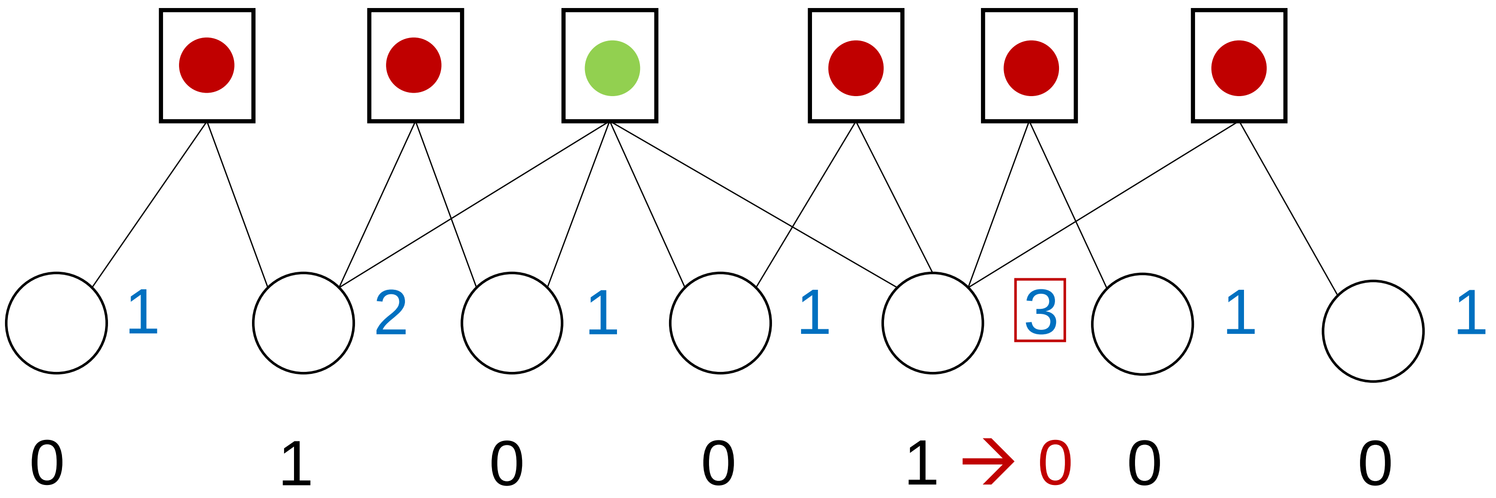
BIKE-1 Decaps

- Bit Flipping Decoding

example

전송 메시지 도착 메시지

X = 0 0 0 0 0 0 0 Y = 0 1 0 0 1 0 0



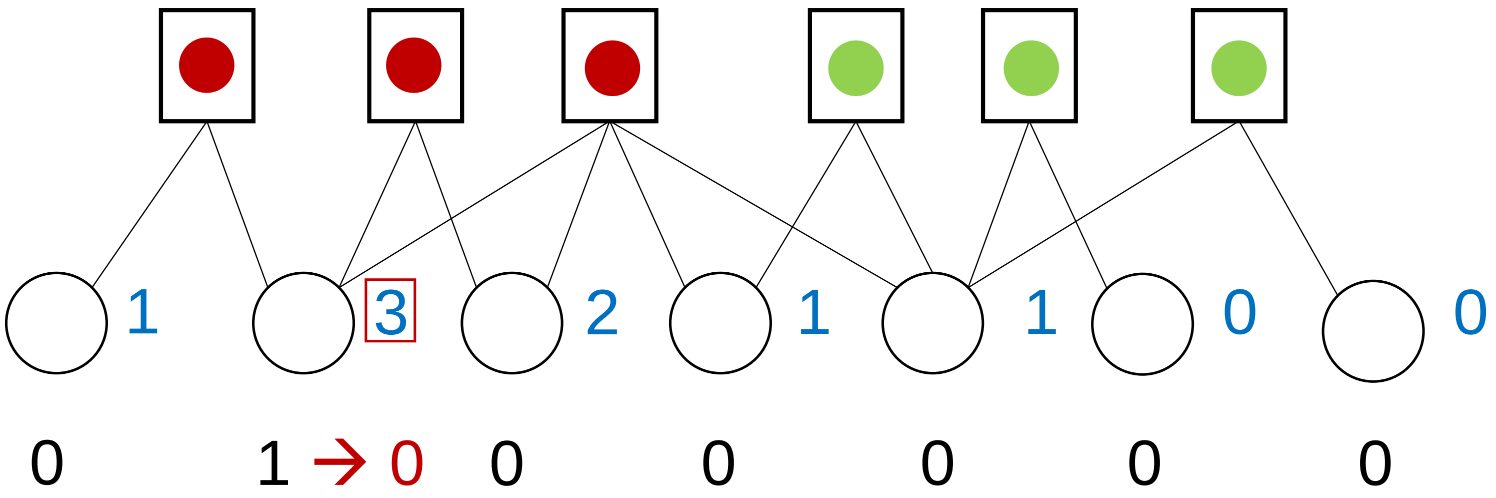
BIKE-1 Decaps

- Bit Flipping Decoding

example

전송 메시지 도착 메시지

X = 0 0 0 0 0 0 0 Y = 0 0 1 0 1 0 0



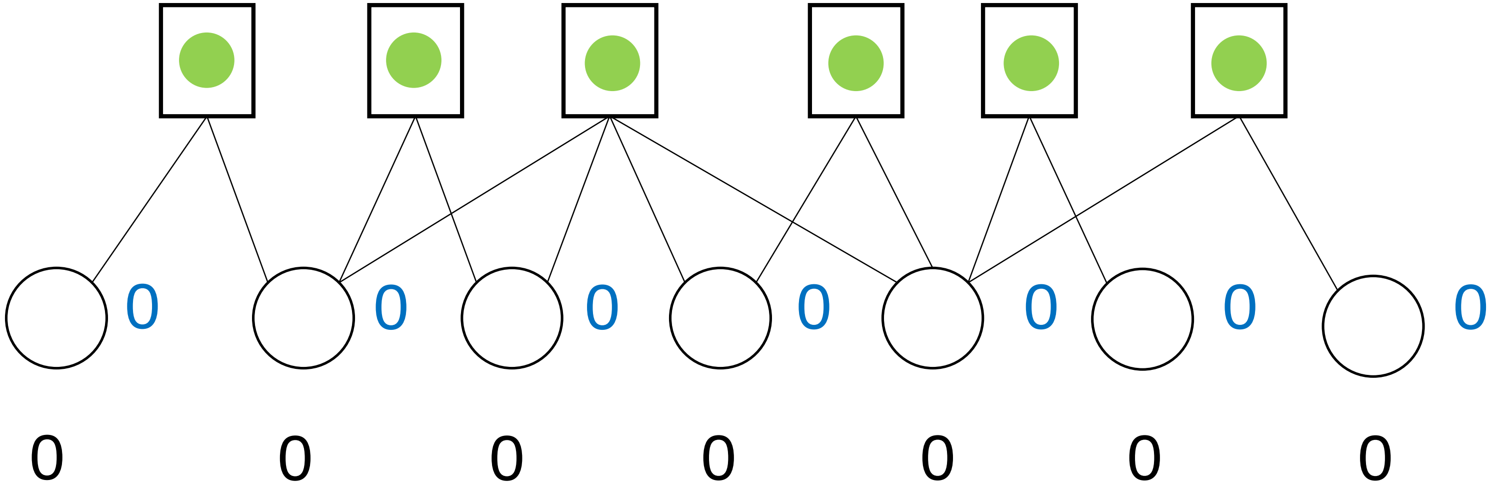
BIKE-1 Decaps

- Bit Flipping Decoding

example

전송 메시지 도착 메시지

X = 0 0 0 0 0 0 0 Y = 0 0 1 0 1 0 0



BIKE-1 Decaps

- Bit Flipping Decoding Algorithm

Require: $H \in \mathbb{F}_2^{(n-k)\times n}$, $s \in \mathbb{F}_2^{n-k}$

Ensure: $eH^T = s$

BIKE-1 Decaps

- Bit Flipping Decoding Algorithm

Require: $H \in \mathbb{F}_2^{(n-k)\times n}$, $s \in \mathbb{F}_2^{n-k}$

Ensure: $eH^T = s$

1: e ← 0
2: s' ← s
3: while s' ≠ 0 do
4: τ ← 미리 정의된 규칙에 의해 결정된 임계값
5: for j = 0,…,n-1 do
6: if $|h_j * s'| \geq \tau |h_j|$ then
7: $e_j \leftarrow e_j + 1 \bmod 2$
8: $s' \leftarrow s - eH^T$
9: return e

$|h_j * s'|$: j를 포함하는 검사되지 않은 패리티 방정식

BIKE-1 Decaps

- Threshold Selection Rule

  Threshold(T)

- $$\pi_1 = \frac{|s| + X}{td} \qquad \pi_0 = \frac{w\,|s| - X}{(n-t)d} \qquad X = \sum_{\ell \text{ odd}} (\ell - 1)\frac{r\binom{w}{\ell}\binom{n-w}{t-\ell}}{\binom{n}{t}}$$

- $$t\binom{d}{T}\pi_1^T(1-\pi_1)^{d-T} \geq (n-t)\binom{d}{T}\pi_0^T(1-\pi_0)^{d-T}$$

- $$T = \left\lceil \frac{\log\frac{n-t}{t} + d\log\frac{1-\pi_0}{1-\pi_1}}{\log\frac{\pi_1}{\pi_0} + \log\frac{1-\pi_0}{1-\pi_1}} \right\rceil$$

BIKE-1 Decaps

- Threshold Selection Rule

  Threshold(T)

BIKE-1, 2

- security level 1: $T = [13.530 + 0.0069722|s|]$
- security level 3: $T = [15.932 + 0.0052936|s|]$
- security level 5: $T = [17.489 + 0.0043536|s|]$

BIKE-3

- security level 1: $T = [13.209 + 0.0060515|s|]$
- security level 3: $T = [15.561 + 0.0046692|s|]$
- security level 5: $T = [17.061 + 0.0038459|s|]$

BIKE 2

- Niederreiter 체계와 패리티 검사 행렬을 사용
- 길이 r의 단일 블록만을 이용함으로서 매우 작은 공식들 형성
- 다항식의 역(Inversion)이 필요함

  - 키 생성과정이 암호화에 비해 느릴 수 있음

BIKE 2

- Niederreiter 체계와 패리티 검사 행렬을 사용
- 길이 r의 단일 블록만을 이용함으로서 매우 작은 공식들 형성
- 다항식의 역(Inversion)이 필요함

  - 키 생성과정이 암호화에 비해 느릴 수 있음
  - 이를 해결하기 위해 집단 키 생성(Batch Key Generation)

    → inverse 연산보다 3번의 곱셈 연산이 더 효율적이다는 가정

ex)
1. 다항식 x와 y 각각 inverse
2. $tmp = xy \rightarrow inv = tmp^{-1} \rightarrow x^{-1} = y \cdot inv$

   $y^{-1} = x \cdot inv$

BIKE-2 KeyGen

- Input: λ , taget quantum security level
- Output: private key(h_0, h_1) and public key *h*

0. λ 가 주어지면 r, w 설정

1. 개인키 h_0, h_1 생성
   - h_0, h_1 무게 = w/2 → 홀수
   - h_0과 h_1은 R로 부터 랜덤하게 선출

BIKE-2 KeyGen

- Input: λ , taget quantum security level
- Output: private key(h_0, h_1) and public key $\boldsymbol{h}$

0. λ 가 주어지면 r, w 설정
1. 개인키 h_0, h_1 생성
2. $\boldsymbol{h} \leftarrow h_1 h_0^{-1}$ 연산

BIKE-2 Encaps

- Input: public key ***h***
- Output: the encapsulated key *K* and the cryptogram c

BIKE-2 Encaps

- Input: public key ***h***
- Output: the encapsulated key *K* and the cryptogram c

1. R^2 공간에서 e_0과 e_1 벡터 선택 ($e_0 + e_1 = t$)

BIKE-2 Encaps

- Input: public key $\boldsymbol{h}$
- Output: the encapsulated key K and the cryptogram c

1. R^2 공간에서 e_0과 e_1 벡터 선택 ($e_0 + e_1 = t$)
2. $c \leftarrow e_0 + e_1\boldsymbol{h}$ 연산

BIKE-2 Encaps

- Input: public key $\boldsymbol{h}$
- Output: the encapsulated key K and the cryptogram c

1. R^2 공간에서 e_0과 e_1 벡터 선택 ($e_0 + e_1 = t$)
2. $c \leftarrow e_0 + e_1\boldsymbol{h}$ 연산
3. $K \leftarrow \mathbf{K}(e_0, e_1)$

   *$\mathbf{K}$: SHA256 해시 함수

BIKE-2 Decaps

- Input: private key h_0, h_1 and cryptogram c
- Output: decapsulated key K or failure symbol ⊥

1. s ← ch_0 연산
2. 에러 백터 e_0', e_1' 을 추출하기 위해 s를 decode
3. 만약 decode 해서 나온 (e_0', e_1')가 t가 안되거나 decoding이 실패하면 실패 신호(⊥**)** 반환 후 정지
4. Decode 성공했다면 나온 e_0'와 e_1'을 갖고 K ← **K**(e_0', e_1') 연산 해서 K 획득

BIKE 3

- BIKE-1과 유사한 점
  - 빠른, Inverse 없는 키 생성
  - 공용 키와 데이터를 위한 두 개의 블록 활용
- Noisy 신드롬에 대한 복호 알고리즘을 사용한다는 점이 차별점

BIKE-3 KeyGen

- Input: λ , taget quantum security level
- Output: private key(h_0, h_1) and public key(f_0, f_1)

0. λ 가 주어지면 r, w 설정
1. 개인키 h_0, h_1 생성
2. g 생성
3. $h_1 + gh_0$, g → f_0, f_1

BIKE-3 Encaps

- Input: public key f_0, f_1
- Output: the encapsulated key *K* and the cryptogram c

1. R^3 공간에서 e, e_0, e_1 벡터 선택 ($e = t/2$, $e_0 + e_1 = t$)
2. $c = (c_0, c_1) \leftarrow (e + e_1 f_0, e_0 + e_1 f_1)$ 연산하여 암호문 생성
3. $K \leftarrow \mathbf{K}(e_0, e_1, e)$ e_0, e_1으로 세션키 생성

   ***K**: SHA256 해시 함수

BIKE-3 Decaps

- Input: private key h_0, h_1 and cryptogram c
- Output: decapsulated key *K* or failure symbol ⊥

1. c를 c0과 c1으로 나누고 신드롬 값 연산 $s \leftarrow c_0 + c_1 h_0$
2. 에러 백터 e_0', e_1', e' 를 추출하기 위해 s를 decode
3. 만약 decode 해서 나온 (e_0', e_1')가 t가 안되고 e가 t/2가 안되거나 decoding이 실패하면실패 신호(⊥**)** 반환 후 정지
4. Decode 성공했다면 나온 e_0', e_1', e'를 갖고 $K \leftarrow \mathbf{K}(e_0', e_1', e')$ 연산 해서 *K* 획득

BIKE-1,2,3 Comparison

	BIKE-1	BIKE-2	BIKE-3
SK	(h_0, h_1) with $\lvert h_0\rvert = \lvert h_1\rvert = w/2$		
PK	$(f_0, f_1) \leftarrow (gh_1, gh_0)$	$(f_0, f_1) \leftarrow (1, h_1h_0^{-1})$	$(f_0, f_1) \leftarrow (h_1 + gh_0, g)$
Enc	$(c_0, c_1) \leftarrow (mf_0 + e_0, mf_1 + e_1)$	$c \leftarrow e_0 + e_1 f_1$	$(c_0, c_1) \leftarrow (e + e_1 f_0, e_0 + e_1 f_1)$
	$K \leftarrow \mathbf{K}(e_0, e_1)$		$K \leftarrow \mathbf{K}(e_0, e_1, e)$
Dec	$s \leftarrow c_0h_0 + c_1h_1 \ ; \ u \leftarrow 0$	$s \leftarrow ch_0 \ ; \ u \leftarrow 0$	$s \leftarrow c_0 + c_1h_0 \ ; \ u \leftarrow t/2$
	$(e_0', e_1') \leftarrow \mathtt{Decode}(s, h_0, h_1, u)$		$(e_0', e_1', e') \leftarrow \mathtt{Decode}(s, h_0, h_1, u)$
	$K \leftarrow \mathbf{K}(e_0', e_1')$		$K \leftarrow \mathbf{K}(e_0', e_1', e')$

BIKE-IND CCA 1,2,3

- 동일한 정적키를 사용하여 여러 번 키 교환할 수 있게 구성

  - Backflip decoder 활용 → 디코딩 실패 경우 감소

  - 실패한 경우들로부터 Key recovery attack을 시행하는 GJS 공격 방지

  *GJS 공격

  디코딩 실패로부터 나온 오류벡터를 관찰해 비밀키의 거리 스펙트럼 계산 후

  거리 스펙트럼을 기반으로 비밀 키를 재구성

*선택암호문공격에 대한 비구별성(Indistinguishability against chosen ciphertext attack; IND-CCA)

BIKE-1-CCA KeyGen

- Input: λ , taget quantum security level
- Output: private key(h_0, $h_{1,}$ $o_{0,}$ o_1) and public key(f_0, f_1)

0. λ 가 주어지면 r, w 설정
1. 개인키 h_0, h_1 생성
2. $o_{0,}$ o_1 생성
3. G 생성
4. gh_1, gh_0 → f_0, f_1

- h_0, h_1 무게 = w/2 → 홀수
- h_0과 h_1은 R로부터 랜덤하게 선출
- o_0과 o_1은 R로부터 랜덤하게 선출
- g는 R로부터 생성

BIKE-1-CCA Encaps

- Input: public key f_0, f_1
- Output: the encapsulated key K and the cryptogram c

1. R에서 랜덤하게 벡터 m 생성
2. $(e_0, e_1) \leftarrow \mathbf{H}(mf_0, mf_1)$
3. $c = (c_0, c_1) \leftarrow (mf_0 + e_0, mf_1 + e_1)$ 연산하여 암호문 생성
4. $K \leftarrow \mathbf{K}(mf_0, mf_{1,}\ e)$으로 세션키 생성

***H**: 해시 함수 ***K**: SHA256 해시 함수

BIKE-1-CCA Decaps

- Input: private key h_0, $h_{1,}$ $o_{0,}$ o_1 and cryptogram c
- Output: decapsulated key K

1. c를 c0과 c1으로 나누고 신드롬 값 연산 s $\leftarrow c_0h_0 + c_1h_1$
2. 에러 백터 e_0', e_1' 을 추출하기 위해 s를 decode(Backflipping Decoding)
3. $(e_0'', e_1'') \leftarrow \mathbf{H}(c_0 + e_0', c_1 + e_1')$
4. 만약 decode 해서 나온 (e_0', e_1')가 t가 안되거나 $(e_0', e_1') != (e_0'', e_1'')$ 이어서 decoding이 실패하면 → $K \leftarrow \mathbf{K}(o_0, o_1, c)$ 연산
5. Decode 성공했다면 $K \leftarrow \mathbf{K}(c_0 + e_0', c_1 + e_1', c)$ 연산 해서 K 획득

BIKE-1 and BIKE-1-CCA Comparison

<table>
<tr><th></th><th>BIKE-1</th><th colspan="2">BIKE-1-CCA</th></tr>
<tr><td rowspan="2">SK</td><td>(h_0, h_1)</td><td colspan="2">$(h_0, h_1, \sigma_0, \sigma_1)$</td></tr>
<tr><td colspan="3">with $|h_0| = |h_1| = w/2$</td></tr>
<tr><td>PK</td><td colspan="3">$(f_0, f_1) \leftarrow (gh_1, gh_0)$</td></tr>
<tr><td rowspan="5">Enc</td><td colspan="3">$m \xleftarrow{\$} \mathcal{R}$</td></tr>
<tr><td>$(e_0, e_1) \xleftarrow{\$} \mathcal{R}^2$</td><td colspan="2">$(e_0, e_1) \leftarrow \mathbf{H}(mf_0, mf_1)$</td></tr>
<tr><td colspan="3">such that $|e_0| + |e_1| = t$</td></tr>
<tr><td colspan="3">$(c_0, c_1) \leftarrow (mf_0 + e_0, mf_1 + e_1)$</td></tr>
<tr><td>$K \leftarrow \mathbf{K}(e_0, e_1, c)$</td><td colspan="2">$K \leftarrow \mathbf{K}(mf_0, mf_1, c)$</td></tr>
<tr><td rowspan="4">Dec</td><td colspan="3">$s \leftarrow c_0 h_0 + c_1 h_1 \ ; \ u \leftarrow 0$</td></tr>
<tr><td colspan="3">$(e'_0, e'_1) \leftarrow \texttt{Decode}(s, h_0, h_1, u)$</td></tr>
<tr><td></td><td colspan="2">$(e''_0, e''_1) \leftarrow \mathbf{H}(c_0 + e'_0, c_1 + e'_1)$</td></tr>
<tr><td>$K \leftarrow \mathbf{K}(e'_0, e'_1, c')$</td><td>$K \leftarrow \mathbf{K}(\sigma_0, \sigma_1, c)$</td><td>$K \leftarrow \mathbf{K}(c_0 + e'_0, c_1 + e'_1, c)$</td></tr>
</table>

Q & A