BIKE-1,2,3

최승주

https://youtu.be/JGWT6dJ-ogQ

HANSUNG UNIVERSITY 한성대학교

Contents

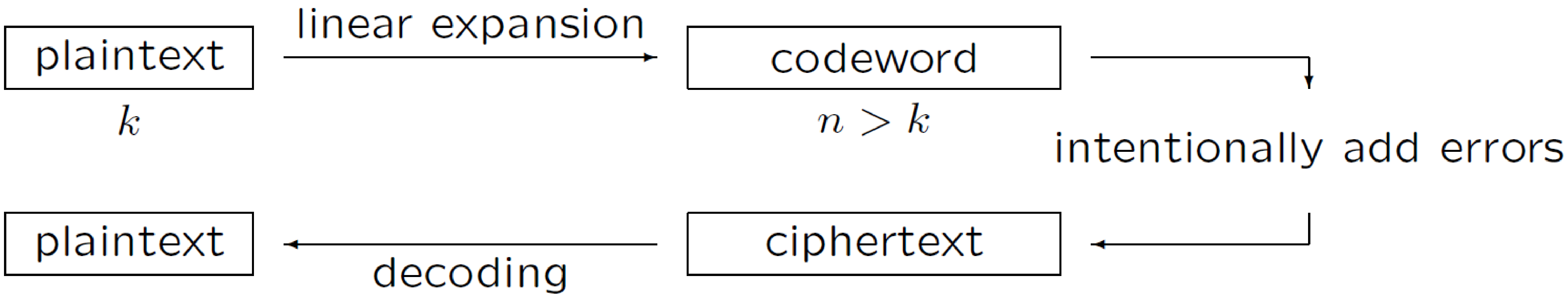
BIKE-1

BIKE-2

BIKE-3

CryptoCraft LAB

부호 이론


plaintext
linear expansion
codeword
k
$n > k$
intentionally add errors
plaintext
ciphertext
decoding

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 e에 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)

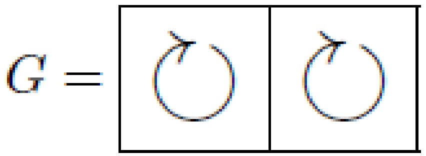
  - 한 행에 들어있는 순환행렬의 개수: index

BIKE – Quasi-Cyclic Codes

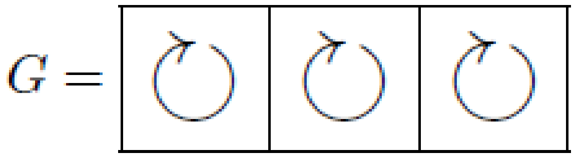
- 준순환부호

  - 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 백터 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

   *$\mathbf{K}$: 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 $\leftarrow c_0h_0 + c_1h_1$
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 $s \leftarrow c_0h_0 + c_1h_1$
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 $s \leftarrow c_0h_0 + c_1h_1$
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
4. Decode 성공했다면 나온 e_0'와 e_1'을 갖고 $K \leftarrow \mathbf{K}(e_0', e_1')$ 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

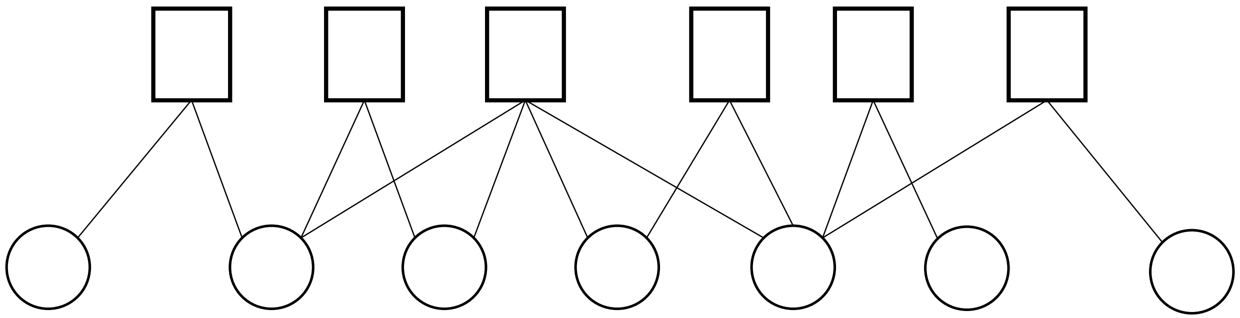
example

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



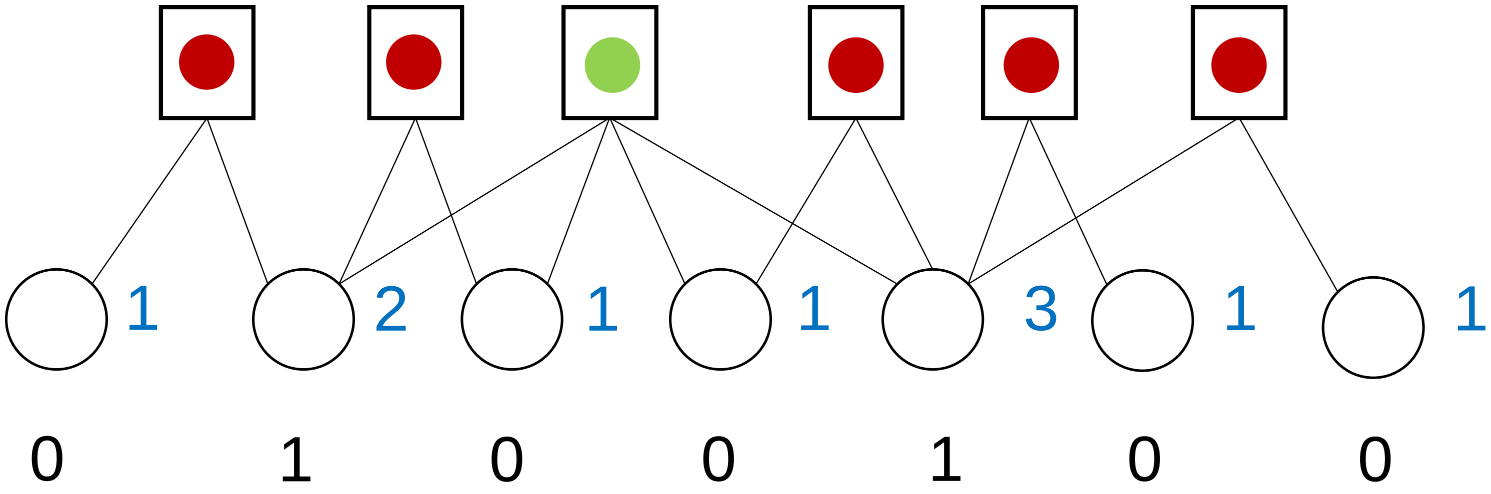
BIKE-1 Decaps

- Bit Flipping Decoding

example

전송 메시지 　　　　　 도착 메시지

X = 0 0 0 0 0 0 0 　 Y = 0 1 0 0 1 0 0



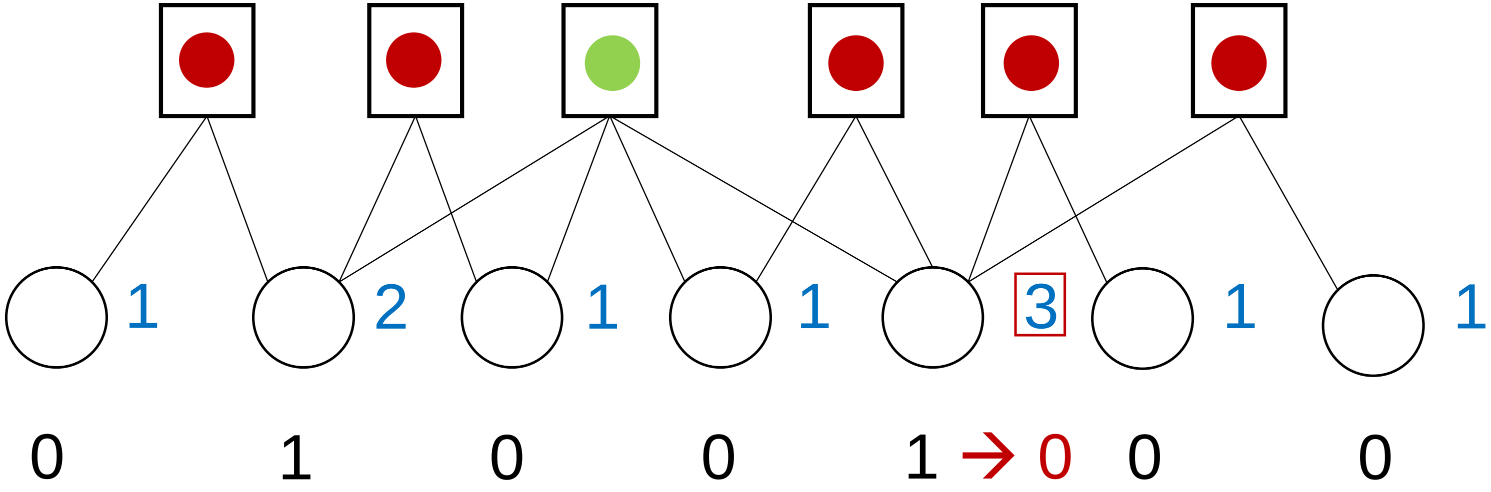
BIKE-1 Decaps

- Bit Flipping Decoding

  example

  전송 메시지 도착 메시지

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



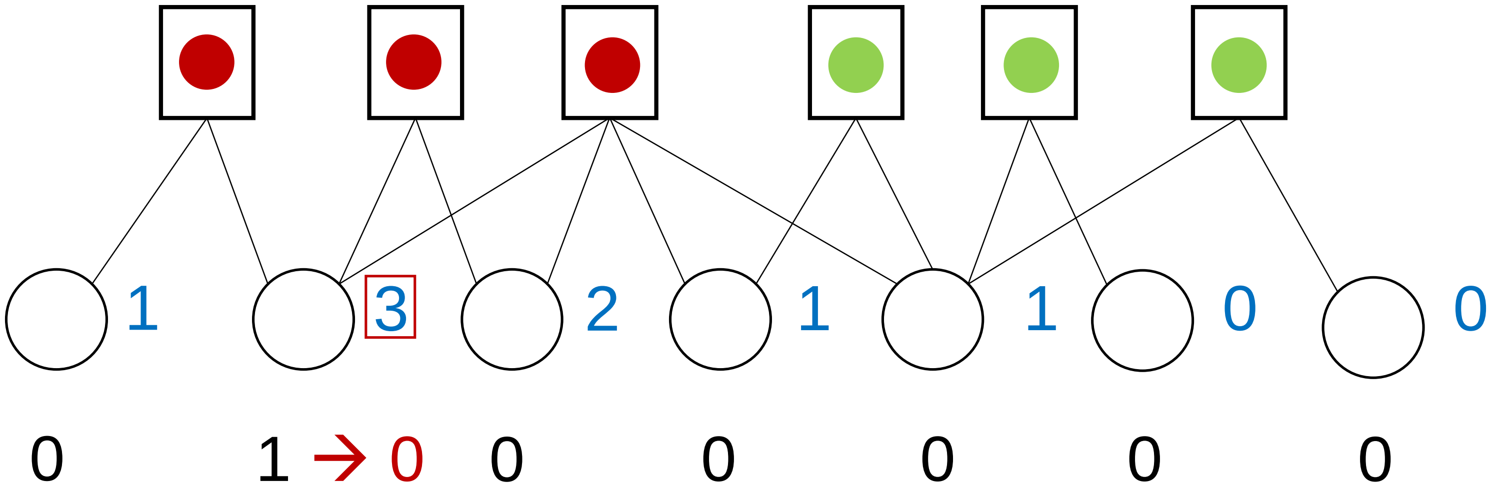
BIKE-1 Decaps

- Bit Flipping Decoding

example

전송 메시지 도착 메시지

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



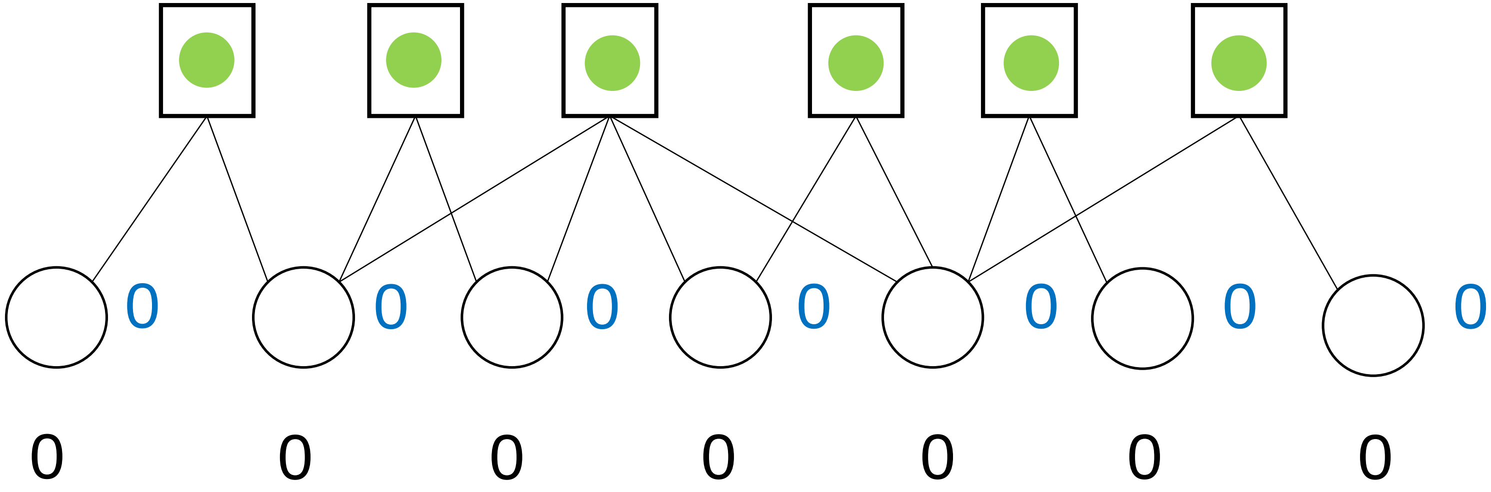
BIKE-1 Decaps

- 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: $eH^T = s$

$e \leftarrow 0$
$s' \leftarrow s$
while $s' \neq 0$ do
  $\tau \leftarrow$ 미리 정의된 규칙에 의해 결정된 임계값
  for $j = 0,\dots,n-1$ do
    if $|h_j * s'| \geq \tau\ |h_j|$ then
      $e_j \leftarrow e_j + 1 \bmod 2$
  $s' \leftarrow s - eH^T$
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: $T = \lceil 13.530 + 0.0069722|s| \rceil$
- security level 3: $T = \lceil 15.932 + 0.0052936|s| \rceil$
- security level 5: $T = \lceil 17.489 + 0.0043536|s| \rceil$

BIKE-3

- security level 1: $T = \lceil 13.209 + 0.0060515|s| \rceil$
- security level 3: $T = \lceil 15.561 + 0.0046692|s| \rceil$
- security level 5: $T = \lceil 17.061 + 0.0038459|s| \rceil$

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

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

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 ***h***

0. λ 가 주어지면 r, w 설정
1. 개인키 h_0, h_1 생성
2. ***h*** ← $h_1 h_0^{-1}$ 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 ***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 ← $e_0 + e_1\boldsymbol{h}$ 연산
3. K ← **K**(e_0, e_1)

*__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 벡터 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

   *$\mathbf{K}$: 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 $\leftarrow$ $c_0 + c_1h_0$
2. 에러 벡터 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* $\leftarrow$ **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	$(f_0, f_1) \leftarrow (gh_1, gh_0)$	$(f_0, f_1) \leftarrow (1, h_1 h_0^{-1})$	$(f_0, f_1) \leftarrow (h_1 + gh_0, g)$
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	$s \leftarrow c_0 h_0 + c_1 h_1$; $u \leftarrow 0$	$s \leftarrow ch_0$; $u \leftarrow 0$	$s \leftarrow c_0 + c_1 h_0$; $u \leftarrow t/2$
	$(e'_0, e'_1) \leftarrow \mathtt{Decode}(s, h_0, h_1, u)$		$(e'_0, e'_1, e') \leftarrow \mathtt{Decode}(s, h_0, h_1, u)$
	$K \leftarrow \mathbf{K}(e'_0, e'_1)$		$K \leftarrow \mathbf{K}(e'_0, e'_1, e')$

NTS-KEM

- McEliece나 Niederreiter 와 같은 공개키 알고리즘의 종류
- NTS-KEM

  최근에는 메시지를 암호화해서 전송하는 쪽보다는 랜덤 키를

  안전하게 전송하는 쪽으로 사용
- McEliece와 마찬가지로 매개변수 3가지 버전을 제공함

  - NIST 요구 사항
- Goppa 코드 사용

Q & A