NIST Round 2 Code-based PQC

Crypto Craft Lab

https://youtu.be/NZHqEPEc0fY

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB
https://crypto.modoo.

Contents

CryptoCraft LAB

Code-based Cryptography

Code-based Cryptography

코딩 이론

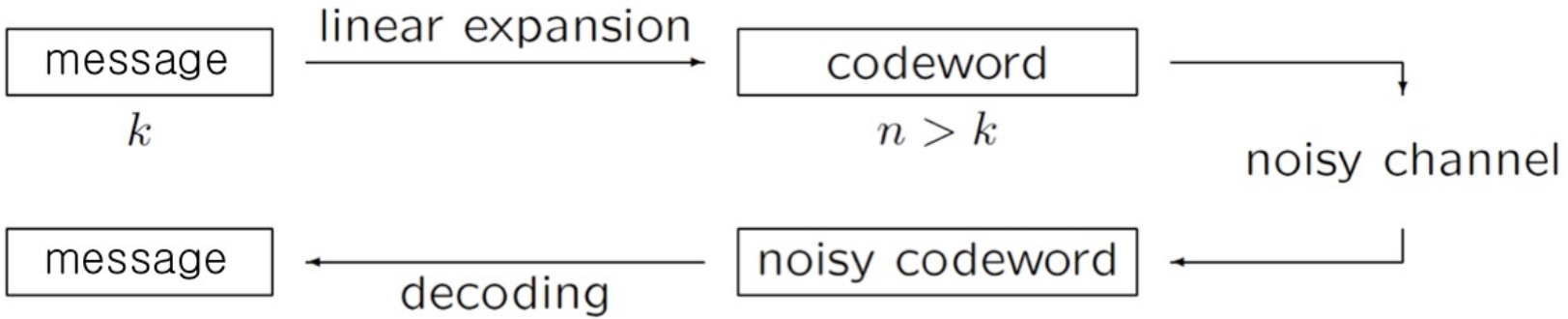
- 1948년 Claude Shannon 제안
- 불완전한 채널(noisy channel)에서 발생하는 잡음 제거
- 코드 기반 암호의 핵심 원리

해밍 코드

선형 부호

코드 기반 암호

코딩 이론 구조



코드 기반 암호 - 해밍 코드

해밍 코드

- 오류 탐지 기능을 추가한 송신 코드

패리티 비트

- 정보 전달 과정에서 오류 발생 여부 확인을 위해 추가되는 비트

패리티 비트 공식

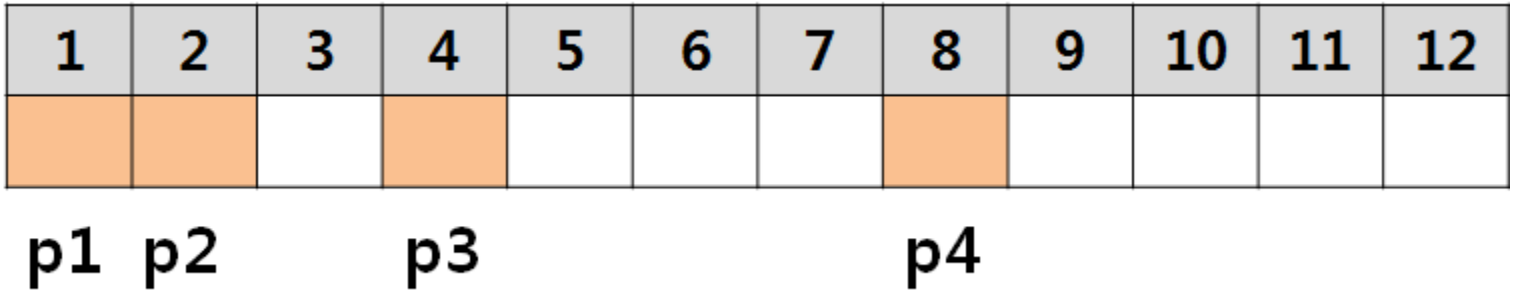
- d: 데이터 비트 수
- p: 패리티 비트

$$2^p+1 \geq d + p$$

코드 기반 암호 - 해밍 코드

패리티 비트

- 1부터 2^n 위치에 배치



코드 기반 암호 - 해밍 코드

ex) (7,4) 해밍 코드

- 4개의 비트 메시지 암호화
- 3개의 패리티 비트 사용

Message bit: x_0 x_1 x_2 x_3

Parity bit: P_1 P_2 P_4

코드 기반 암호 - 해밍 코드

(7,4) 해밍 코드

Message bit: x_0 x_1 x_2 x_3

Parity bit: P_1 P_2 P_4

Hamming code structure

1	2	3	4	5	6	7
p_1	p_2	x_3	p_4	x_2	x_1	x_0

코드 기반 암호 - 해밍 코드

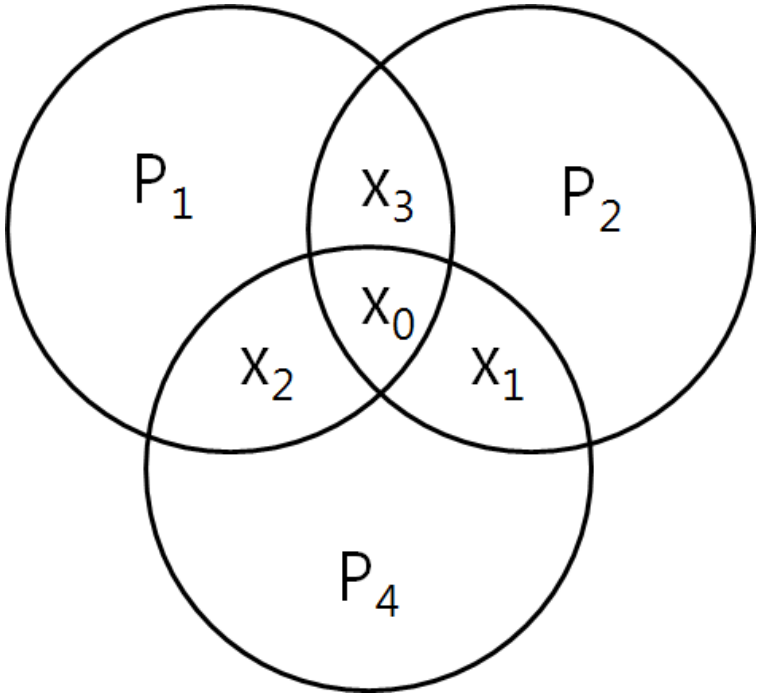
(7,4) 해밍 코드

Message bit: x_0 x_1 x_2 x_3

Parity bit: P_1 P_2 P_4

Hamming code structure

1	2	3	4	5	6	7
p_1	p_2	x_3	p_4	x_2	x_1	x_0



코드 기반 암호 - 해밍 코드

Hamming code structure

1	2	3	4	5	6	7
p_1	p_2	x_3	p_4	x_2	x_1	x_0

p1 1, 3, 5, 7 담당

P1 = X3 ⊕ x2 ⊕ x0

1	001
3	011
5	101
7	111

p2 2, 3, 6, 7 담당

P2 = X3 ⊕ x1 ⊕ x0

2	010
3	011
6	110
7	111

p4 4, 5, 6, 7 담당

P4 = X2 ⊕ x1 ⊕ x0

4	100
5	101
6	110
7	111

코드 기반 암호 - 해밍 코드

Hamming code structure

1	2	3	4	5	6	7
p_1	p_2	x_3	p_4	x_2	x_1	x_0

에러 체크

C1 = P1 ⊕ X3 ⊕ x2 ⊕ x0

C2 = P2 ⊕ X3 ⊕ x1 ⊕ x0

C4 = P4 ⊕ X2 ⊕ x1 ⊕ x0

→ 연산 결과가 0일시 정상

코드 기반 암호 - 해밍 코드

에러 체크

ex)

Hamming code example

1	2	3	4	5	6	7
p_1	p_2	x_3	p_4	x_2	x_1	x_0
1	1	0	0	1	1	0

1	0	0	0	1	1	0

코드 기반 암호 - 해밍 코드

에러 체크

Hamming code example

1	2	3	4	5	6	7
p_1	p_2	x_3	p_4	x_2	x_1	x_0
1	0	0	0	1	1	0

C1 = P1 ⊕ X3 ⊕ x2 ⊕ x0 = 0

C2 = P2 ⊕ X3 ⊕ x1 ⊕ x0 = 1

C4 = P4 ⊕ X2 ⊕ x1 ⊕ x0 = 0

C4 C2 C1

0 1 0 = 2 → 2번째 자리에 에러

코드 기반 암호 - 선형 부호

선형 부호(Linear code)

- [n k] 코드를 갖고 진행
- n: 코드의 길이
- k: 차원

코드 기반 암호 - 선형 부호

$$G(\text{Generating Matrix}) = \overbrace{\left[\, I_k \mid P \,\right]}^{n}\Big\}_{k}$$

$$H(\text{Pairty Check Matrix}) = \underbrace{\left[\, -P^T \mid I_{n-k} \,\right]}_{n}\Big\}_{n-k}$$

I_k : 단위 행렬

P : 에러 수정 행렬

코드 기반 암호 - 선형 부호

G와 H 사이를 자유롭게 오갈 수(변경) 있음

ex)

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} I_k \mid P \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}_{\substack{k=2 \\ n=4}}$$

$$H = \begin{bmatrix} -P^T \mid I_{n-k} \end{bmatrix} = \begin{bmatrix} 0 & -1 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

T = 전위

코드 기반 암호 - 선형 부호

생성된 G와 H^T 곱은 0

$$G \qquad H^T$$

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

코드 기반 암호 - McEliece

- 1978, Robert J. McEliece에 의해 제안
- 대표적인 부호 기반(Code-based) 암호
- 양자 컴퓨터 상에서도 안전하다고 여겨지는 암호
- 양자 컴퓨터 시대에 대비하는 움직임과 함께 다시 주목을 받음
- Goppa 코드(부호) 공간 활용

코드 기반 암호 - McEliece

→ 키생성 : 주어진 파라미터를 이용한다.

G : 최소거리 $(d \geq 2t+1)$ 인 이진부호 g에대한 $k \times n$ 생성행렬 *Goppa* 부호

S : 임의의 $k \times k$ 이진 정칙 행렬

P : 임의의 $n \times n$ 순열행렬

G' : SGP

공개키 : (G', t)

비밀키 : $\left(S, D_g, P\right)$ 이때 D_g는 g에 대한 효율적인 복호 알고리즘이다.

→ 암호화: 암호문 c는 이렇게 생성된다.

$$c = mG' \oplus e$$

→ 복호화 : 먼저 $cP^{-1} = (mS)G \oplus eP^{-1}$를 계산한 후, 복호알고리즘 D_g을 적용하여 다음과 같이 계산한다.

$$mSG = D_g\left(cP^{-1}\right)$$

복호알고리즘 후 $m = mS \cdot S^{-1}$을 계산하면

원본 메세지인 m 을 획득할 수 있다.

McEliece 암호화 - Key generation

개인키

- 비밀키로 사용되는 generator matrix(***G***)는 랜덤한 **Goppa 코드** 공간에서 뽑아냄

  Ex) [7, 4, 3]

- *k* x *k* 의 가역행렬 S와 *n* x *n* 의 순열행렬 *P*를 랜덤하게 선택

$$G = \begin{matrix} 1\,0\,0\,0\,1\,1\,0 \\ 0\,1\,0\,0\,1\,0\,1 \\ 0\,0\,1\,0\,0\,1\,1 \\ 0\,0\,0\,1\,1\,1\,1 \end{matrix} \qquad S = \begin{matrix} 1\,1\,0\,1 \\ 1\,0\,0\,1 \\ 0\,1\,1\,1 \\ 1\,1\,0\,0 \end{matrix} \qquad P = \begin{matrix} 0\,1\,0\,0\,0\,0\,0 \\ 0\,0\,0\,1\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,1 \\ 1\,0\,0\,0\,0\,0\,0 \\ 0\,0\,1\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0\,1\,0 \\ 0\,0\,0\,0\,1\,0\,0 \end{matrix}$$

공개키

- 개인키 ***G, S, P*** 를 사용하여 공개키(***G'***) 생성

  → G를 숨기기 위해서 S와 P를 이용하여 숨김

$$G' = S \cdot G \cdot P = \begin{matrix} 1\,1\,1\,1\,0\,0\,0 \\ 1\,1\,0\,0\,1\,0\,0 \\ 1\,0\,0\,1\,1\,0\,1 \\ 0\,1\,0\,1\,1\,1\,0 \end{matrix}$$

McEliece 암호화 - Encoding & Encryption

- k-bit의 메시지 ***m***을 암호화

  $c = m \cdot G' \oplus e$

  $m = 1\ 1\ 0\ 1$

  $e = 0\ 0\ 0\ 0\ 1\ 0\ 0$

  $c = m \cdot G' \oplus e = 0\ 1\ 1\ 0\ 0\ 1\ 0 \oplus 0\ 0\ 0\ 0\ 1\ 0\ 0$

  $= 0\ 1\ 1\ 0\ 1\ 1\ 0$

$C = 0\ 1\ 1\ 0\ 1\ 1\ 0$

- 송신자는 암호문(C) 전송

  - e는 weight t인 길이 n-bit 오류 벡터

McEliece 복호화 - Decoding & Decryption

- 수신자는 P의 역행렬을 암호문 C 우측에 곱셈 연산 진행

$$cP^{-1} = mSG \oplus eP^{-1} = 1\ 0\ 0\ 0\ 1\ 1\ 1$$

- ***P*** 는 순열행렬이므로 오류벡터 e의 weight를 변경시키지 않음
  - 포함되어 있는 오류가 변경되지 않음(순서만 섞임)

- G에 해당하는 Parrity matrix(***H***)를 사용한 **Syndrome Decoding** 과정 수행
  - Syndrome: Codeword에 해당하는 패리티 검사 행렬(H)을 계산한 값
  - 오류가 없으면 Syndrome 값은 0을 반환

McEliece 복호화 - Decoding & Decryption

- $cP^{-1} = mSG \oplus eP^{-1} = 1\ 0\ 0\ 0\ 1\ 1\ 1$ ➔ 신드롬 값 계산

$$Hc'^{t} = H(msG \oplus eP^{-1})^{T} = H(msG)^{T} \oplus H(eP^{-1})^{T} = H(eP^{-1})^{T}$$

$$\underset{G}{\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}} \cdot \underset{H^{T}}{\begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$G = \left[I_k \mid P \right] \quad H = \left[-P^{T} \mid I_{n-k} \right]$$

McEliece 복호화 - Decoding & Decryption

- G에 해당하는 Parrity check matrix(*H*)는 수신자만 보유

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

$$cP^{-1} = mSG \oplus eP^{-1} = 1\,0\,0\,0\,1\,1\,1 \cdot H^{T} = 0\,0\,1$$

$$cP^{-1} = mSG \rightarrow 1\,0\,0\,0\,1\,1\,0$$

McEliece 복호화 - Decoding & Decryption

- $cP^{-1} = mSG =$ 1 0 0 0 1 1 0

  - 오류가 수정된 메시지

- G의 구조적 특성을 활용해 mS 도출

$$mS = 1\,0\,0\,0 \qquad G = \begin{bmatrix} 1\,0\,0\,0\,1\,1\,0 \\ 0\,1\,0\,0\,1\,0\,1 \\ 0\,0\,1\,0\,0\,1\,1 \\ 0\,0\,0\,1\,1\,1\,1 \end{bmatrix} \qquad G = \left[\, I_k \mid P \,\right]$$

McEliece 복호화 - Decoding & Decryption

- $mS = 1\ 0\ 0\ 0$
- S는 가역행렬
- S^{-1}을 우측에 곱하여 원본 메시지 m을 획득

$$S^{-1} = \begin{matrix} 1\ 1\ 0\ 1 \\ 1\ 1\ 0\ 0 \\ 0\ 1\ 1\ 1 \\ 1\ 0\ 0\ 1 \end{matrix} = mS \cdot S^{-1} = 1\ 1\ 0\ 1$$

$$m = 1\ 1\ 0\ 1$$

코드 기반 암호 - 선형 이진 부호

Parameter: [*n, k, d*]

minimum weight가 *d*인 선형 이진 후보 [***n, k***]는 최대 $t = \frac{d-1}{2}$ 의 오류를 수정할 수 있음

Definition:

[***n, k***] 는 F_2에서의 **k** 선형 독립한 벡터들로 표현 가능

해당 ***k*** 벡터들을 ***k*** x ***n***의 행렬로 사용 → Generator Matrix(***G***)

코드 내의 모든 벡터들을 generator matrix 행들의 선형조합으로 생성 가능

코드 기반 암호 - 이진 선형 부호

- 길이 ***k*** 의 메시지 ***m*** 을 암호화 하기 위해 Goppa code ***G*** 를 사용하여 길이 ***n*** 으로 선형확장 → 인코딩

(Message) x [Goppa code] = (codeword)

(***1*** x ***k***) (***k*** x ***n***) (***1*** x ***n***)

선형확장

(Linear expansion)

- 디코딩 과정에서는 ***G*** 에 해당하는 Parrity matrix ***H*** 가 사용됨 → Syndrome Decoding

codeword x [Parrity Check matrix.] = s, 오류가 없다면 s = 0

(***1*** x ***n***) (***n*** − ***k*** x ***n***)

코드 기반 암호 - 이진 선형 부호

- 해당 codeword ***c*** 에 오류가 추가 되어도 수정 가능
  - Goppa code가 오류수정 역할을 수행 → 공개키로 사용된다.
  - 송신자들은 자신의 메시지와 Goppa code를 사용하여 codeword를 생성, 그 뒤에 오류 ***e*** 를 임의로 추가하여 원본 메세지를 암호화 → ***mG*** + ***e*** = ***codeword*** (암호문)

- 해당 Goppa code (***G***)를 그대로 공개키로 사용하면 누구나 오류를 수정할 수 있음
  → ***G*** 를 숨기는 과정이 존재

- 수신자는 ***G*** 를 활용하여 수신된 암호문의 오류를 수정(Syndrome decoding)하여 원본 메세지를 획득

Goppa & Information Set Decoding

Definition of a Goppa Code

Goppa 코드 : $\Gamma(L, g(z))$

갈루아 필드 $GF(q^m)$ 상의 t 차 다항식 $g(z)$, $GF(q^m)$ 의 subset L 에 의해 정의된다.
(q는 소수)

$$g(z) = g_0 + g_1 z + \ldots + g_t z^t = \sum_{i=0}^{t} g_i z^i,$$
$$L = \{\alpha_1, \ldots, \alpha_n\} \subseteq GF(q^m),$$

$g(\alpha_i) \neq 0$ 인 모든 $\alpha_i \in L$ 와 $GF(q)$ 상 벡터 $c = (c_1, \ldots, c_n)$ 로 다음 함수를 뽑아낸다.

$$R_c(z) = \sum_{i=1}^{n} \frac{c_i}{z - \alpha_i},$$

Definition of a Goppa Code

$$g(z) = g_0 + g_1 z + \ldots + g_t z^t = \sum_{i=0}^{t} g_i z^i,$$
$$L = \{\alpha_1, \ldots, \alpha_n\} \subseteq GF(q^m),$$
$$R_c(z) = \sum_{i=1}^{n} \frac{c_i}{z - \alpha_i},$$

Definition. Goppa 코드 $\Gamma(L, g(z))$ 는 아래를 만족하는 모든 벡터들 c로 구성된다.

$$R_c(z) \equiv 0 \pmod{g(z)}.$$

* ≡ → 합동

Parameters of a Goppa Code

여기서 $\frac{1}{z-a_i}$ 은 다항식 $p_i(z)$ modulo $g(z)$ 로 표현할 수 있다.

$$\frac{1}{z-\alpha_i} \equiv p_i(z) = p_{i1} + p_{i2}z + \ldots + p_{it}z^{t-1} \pmod{g(z)}.$$

*

$$\frac{1}{z-a_i} \equiv \frac{p(z)}{g(z)+1} \quad \text{modulo } g(z)$$

$$g(z)+1 \equiv p(z)(z-a_i) \quad \text{modulo } g(z)$$

$$\frac{1}{z-a_i} \equiv p(z) \quad \text{modulo } g(z)$$

Parameters of a Goppa Code

$$\frac{1}{z-\alpha_i} \equiv p_i(z) = p_{i1} + p_{i2}z + \ldots + p_{it}z^{t-1} \pmod{g(z)}.$$

* $$R_c(z) = \sum_{i=1}^{n} \frac{c_i}{z-\alpha_i},$$

그러므로 식 $R_c(z) \equiv 0 \pmod{g(z)}$. 는 다음과 같이 쓸 수 있다.

$$\sum_{i=1}^{n} c_i p_i(z) \equiv 0 \pmod{(g(z))}.$$

그리고 z^j 의 계수를 분리해서 정리하면 다음과 같이 쓸 수 있다.

$$\sum_{i=1}^{n} c_i p_{ij} = 0, \text{ for } 1 \leq j \leq t.$$

Parity Check Matrix of the Goppa Code

디코딩을 하기 위해서는 우선, 패리티 체크 행렬 H 가 필요 *

앞서, 코드워드 $c = (c_1, \ldots, c_n)$ 는

$\frac{1}{z-\alpha_i} \equiv p_{i1} + p_{i2}z + \ldots + p_{it}z^{t-1} \pmod{(g(z))}$ 의 p_{ij} 에 대하여 다음을 만족해야 하는 것을 보였음

$$\sum_{i=1}^{n} c_i p_{ij} = 0, \text{ for } 1 \leq j \leq t.$$

패리티 체크 행렬 H 는 코드워드에 c 대하여 $cH^T = 0$ 을 만족해야 한다. 그러므로 H 는 다음과 같다.

$$H = \begin{pmatrix} p_{11} & \cdots & p_{n1} \\ \vdots & \ddots & \vdots \\ p_{1t} & \cdots & p_{nt} \end{pmatrix}$$

Parity Check Matrix of the Goppa Code

H의 요소 p_{ij} 를 구하기 위해서 $p_i(z)$ 다시 표현

$$p_i(z) \equiv (z-\alpha_i)^{-1} \equiv -\frac{g(z)-g(\alpha_i)}{z-\alpha_i} \cdot g(\alpha_i)^{-1}$$

* $$H = \begin{pmatrix} p_{11} & \cdots & p_{n1} \\ \vdots & \ddots & \vdots \\ p_{1t} & \cdots & p_{nt} \end{pmatrix}$$

이는 $(z-\alpha_i)$의 곱으로 확인해 볼 수 있다.

이제 $h_i := g(\alpha_i)^{-1}$ 로 정의하고 앞서 $g(z) = g_0 + g_1 z + \ldots + g_t z^t$ 였다.
이걸로 다음 식을 사용한다.

$$p_i(z) = -\frac{g_t \cdot (z^t - \alpha_i^t) + \ldots + g_1 \cdot (z-\alpha_i)}{z-\alpha_i} \cdot h_i.$$

위의 분수식은 다음과 같이 다시 쓰일 수 있다.

$$g_t(z^{t-1}+z^{t-2}\alpha_i+\ldots+\alpha_i^{t-1})+g_{t-1}(z^{t-2}+z^{t-3}\alpha_i+\ldots+\alpha_i^{t-2})+\ldots+g_2(z+\alpha_i)+g_1$$

Parity Check Matrix of the Goppa Code

$$\{\, g_t(z^{t-1}+z^{t-2}\alpha_i+\ldots+\alpha_i^{t-1})+g_{t-1}(z^{t-2}+z^{t-3}\alpha_i+\ldots+\alpha_i^{t-2})+\ldots+g_2(z+\alpha_i)+g_1 \,\} \cdot h_i.$$

이제 $p_i(z) = p_{i1} + p_{i2}z + \ldots + p_{it}z^{t-1}$ 를 기반으로 p_{ij}에 대한 다음과 같은 표현을 찾는다.

$$\begin{cases} p_{i1} &= -(g_t\alpha_i^{t-1} + g_{t-1}\alpha_i^{t-2} + \ldots + g_2\alpha_i + g_1)h_i; \\ p_{i2} &= -(g_t\alpha_i^{t-2} + g_{t-1}\alpha_i^{t-3} + \ldots + g_2)h_i; \\ &\vdots \\ p_{i(t-1)} &= -(g_t\alpha_i + g_{t-1})h_i; \\ p_{it} &= -g_t h_i. \end{cases}$$

Parity Check Matrix of the Goppa Code

$$H = \begin{pmatrix} p_{11} & \dots & p_{n1} \\ \vdots & \ddots & \vdots \\ p_{1t} & \dots & p_{nt} \end{pmatrix}$$ 와

$$\begin{cases} p_{i1} &= -(g_t\alpha_i^{t-1} + g_{t-1}\alpha_i^{t-2} + \ldots + g_2\alpha_i + g_1)h_i; \\ p_{i2} &= -(g_t\alpha_i^{t-2} + g_{t-1}\alpha_i^{t-3} + \ldots + g_2)h_i; \\ & \vdots \\ p_{i(t-1)} &= -(g_t\alpha_i + g_{t-1})h_i; \\ p_{it} &= -g_t h_i. \end{cases}$$ 로 부터

$H = CXY$ 를 찾을 수 있음

$$C = \begin{pmatrix} -g_t & -g_{t-1} & -g_{t-2} & \dots & -g_1 \\ 0 & -g_t & -g_{t-1} & \dots & -g_2 \\ 0 & 0 & -g_t & \dots & -g_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & -g_t \end{pmatrix},$$

$$X = \begin{pmatrix} \alpha_1^{t-1} & \alpha_2^{t-1} & \dots & \alpha_n^{t-1} \\ \alpha_1^{t-2} & \alpha_2^{t-2} & \dots & \alpha_n^{t-2} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1 & \alpha_2 & \dots & \alpha_n \\ 1 & 1 & \dots & 1 \end{pmatrix}, \quad Y = \begin{pmatrix} h_1 & 0 & 0 & \dots & 0 \\ 0 & h_2 & 0 & \dots & 0 \\ 0 & 0 & h_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & h_n \end{pmatrix}$$

Generator Matrix of the Goppa Code

패리티 체크 행렬 *H* 는 오류를 수정하기 위해 사용되고

암호시스템엔 메세지를 인코딩하고 디코딩 할 생성행렬(Generator Matrix)이 필요.

코드워드 *c* 는 메세지 $m = (m_1, \ldots, m_k)$ 과 생성행렬 *G* 의 곱으로 형성된다. 후에 코드워드의 오류는

모든 $c \in \Gamma(L, g(z))$ 에 대하여 $cH^T = 0$ 의 성질을 이용하여 수정된다. 그러므로 *c* 로 구성되는 *G* 는 다음과 같고

$$GH^T = 0$$

H 로부터 *G* 를 구할 수 있다.

Generate Parrity Check Matrix from the Goppa Code

$g(z) = z^2 - 1$ 그리고, $L = \{ a^i \mid 1 \le i \le 9 \} \subseteq GF(2^4)$ 를 사용한다. * 참고로 L에 사용될 후보 군은 매우 많다.

이제 q = 2, m = 4, n = 9, t = 2 의 Goppa 코드를 가지게 되는 것이다. 그리고 앞서 말한 특성으로

$k \geq 9 - 4 \cdot 2 = 1$ and $d \geq 2 + 1 = 3$ 이기 때문에 명칭으로 [9, ≥ 1, ≥ 3] Goppa 이다.

$h_i := g(\alpha_i)^{-1}$ 와

$$\begin{cases} p_{i1} &= -(g_t\alpha_i^{t-1} + g_{t-1}\alpha_i^{t-2} + \ldots + g_2\alpha_i + g_1)h_i; \\ p_{i2} &= -(g_t\alpha_i^{t-2} + g_{t-1}\alpha_i^{t-3} + \ldots + g_2)h_i; \\ &\vdots \\ p_{i(t-1)} &= -(g_t\alpha_i + g_{t-1})h_i; \\ p_{it} &= -g_t h_i. \end{cases}$$

로 부터 다음 H 를 찾아 낸다

$$H = \begin{pmatrix} \alpha h_1 & \alpha^2 h_2 & \ldots & \alpha^9 h_9 \\ h_1 & h_2 & \ldots & h_9 \end{pmatrix}$$

* $H = \begin{pmatrix} p_{11} & \ldots & p_{n1} \\ \vdots & \ddots & \vdots \\ p_{1t} & \ldots & p_{nt} \end{pmatrix}$ $g_0 = 1$, $g_1 = 0$, $g_2 = 1$

Generate Parrity Check Matrix from the Goppa Code

Therefore, $GF(2^4)^* = < \alpha >$, or equivalently,

$$GF(2^4) = \{0, \alpha, \alpha^2, \ \ldots \ , \alpha^{14}\}.$$

We represent the elements of $GF(2^4)^*$ as the powers of α, using $\alpha^4 = \alpha + 1$.
Of course, we represent the element 0 as $(0,0,0,0)^T$.

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

Generate Parrity Check Matrix from the Goppa Code

$$H = \begin{pmatrix} \alpha h_1 & \alpha^2 h_2 & \dots & \alpha^9 h_9 \\ h_1 & h_2 & \dots & h_9 \end{pmatrix}$$

$$H = \begin{pmatrix} \alpha^8 & \alpha & \alpha^5 & \alpha^2 & 1 & \alpha^{10} & \alpha^4 & \alpha^4 & \alpha^{10} \\ \alpha^7 & \alpha^{14} & \alpha^2 & \alpha^{13} & \alpha^{10} & \alpha^4 & \alpha^{12} & \alpha^{11} & \alpha \end{pmatrix}$$

앞서 언급한

Parity Check Matrix of the Goppa Code

디코딩을 하기 위해서는 우선, 패리티 체크 행렬 *H*가 필요 *

앞서, 코드워드 $c = (c_1, \dots, c_n)$ 는

$\frac{1}{z-\alpha_i} \equiv p_{i1} + p_{i2}z + \dots + p_{it}z^{t-1} \pmod{(g(z))}$ 의 p_{ij} 에 대하여 다음을 만족해야 하는 것을 보였음

$$\sum_{i=1}^{n} c_i p_{ij} = 0, \text{ for } 1 \le j \le t.$$

*

$$h_i := g(\alpha_i)^{-1}$$

$g(z) = z^2 - 1$

$h_i = (a_i{}^2 - 1)^{-1}$

$\frac{1}{z-\alpha^9} \equiv \alpha^{10} + \alpha z \pmod{z^2 - 1}$ (9번째 컬럼) 을 검증해보면

*

$a^{19} = a^4$

Generate Parrity Check Matrix from the Goppa Code

$$H = \begin{pmatrix} \alpha^8 & \alpha & \alpha^5 & \alpha^2 & 1 & \alpha^{10} & \alpha^4 & \alpha^4 & \alpha^{10} \\ \alpha^7 & \alpha^{14} & \alpha^2 & \alpha^{13} & \alpha^{10} & \alpha^4 & \alpha^{12} & \alpha^{11} & \alpha \end{pmatrix}$$ 로 부터 binary 형식의 *H* 를 표현 할 수 있다.

그리고 *H* 를 구함으로써 생성행렬인 *G* 도 구할 수 있다. 그 결과, 이것이 [9, ≥ 1, ≥ 3] Goppa Code 가 된다.

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

Encoding & Correcting Errors

$\Gamma(L, g(z))$: 갈루아 필드 *GF*(q^m) 상의 *t* 차 다항식 $g(z)$, *GF*(q^m) 의 subset L 에 의해 정의된

k dimension, size *n*, minimum distance *d*. Goppa 코드에서 메세지는 다음과 같이 인코딩 된다.

$$(m_1, \ \ldots \ , m_k) \cdot G = (c_1, \ \ldots \ , c_n)$$

y 가 *r* 개의 error 를 수신한 메세지라 하면 ($r \leq \frac{d-1}{2}$)

$$(y_1, \ \ldots \ , y_n) = (c_1, \ \ldots \ , c_n) + (e_1, \ \ldots \ , e_n)$$

r 위치의 $e_i \neq 0$, 이제 오류를 수정하기 위해 오류 벡터 e를 찾아내야 한다. 그러므로 다음을 찾아내야 한다.

- 오류 위치의 그룹 $B = \{i \mid e_i \neq 0\}$
- 해당 오류 값 $e_i \text{ for } i \in B$

Encoding & Correcting Errors

이를 찾기 위해, 두가지 다항식을 정의 한다.

- $\sigma(z) := \prod_{i \in B}(z - \alpha_i)$ → 오류 위치 다항식
- $\omega(z) := \sum_{i \in B} e_i \prod_{j \in B, j \neq i}(z - \alpha_j)$ → 오류 평가 다항식

이 다항식들과 신드롬 *s(z)* 과의 상관관계를 사용하여 수신한 메세지의 오류를 수정할 수 있다.

$$\begin{aligned} s(z) &:= \sum_{i=1}^{n} \frac{y_i}{z - \alpha_i} = \sum_{i=1}^{n} \frac{c_i + e_i}{z - \alpha_i} = \sum_{i=1}^{n} \frac{c_i}{z - \alpha_i} + \sum_{i \in B} \frac{e_i}{z - \alpha_i} \\ &\equiv \sum_{i \in B} \frac{e_i}{z - \alpha_i} \pmod{g(z)}. \end{aligned}$$

Encoding & Correcting Errors

weight *r* 의 오류 벡터 에서의 $\sigma(z), \omega(z)\ s(z)$ 에서 다음과 같은 특성이 발견된다.

1. $\deg(\sigma(z)) = r;$
2. $\deg(\omega(z)) \leq r - 1;$

5. $\sigma(z)s(z) \equiv \omega(z) \pmod{g(z)}.$

* 1,2 증명

- $\sigma(z) := \prod_{i \in B}(z - \alpha_i)$
- $\omega(z) := \sum_{i \in B} e_i \prod_{j \in B, j \neq i}(z - \alpha_j)$

* 5 증명

$$\begin{aligned}\sigma(z)s(z) &\equiv \prod_{i \in B}(z - \alpha_i) \sum_{i \in B} \frac{e_i}{z - \alpha_i} \\ &= \sum_{i \in B} e_i \prod_{j \in B, j \neq i}(z - \alpha_j) \\ &= \omega(z).\end{aligned}$$

Encoding & Correcting Errors

코드워드에서 오류를 수정하기 위한 핵심 방정식은 $\sigma(z)s(z) \equiv \omega(z) \pmod{g(z)}$

g(z) 는 알고 있고, *s(z)* 도 계산 가능하기 때문에, 우리가 알아내야 할 식은

$$\sigma(z) = \sigma_0 + \sigma_1 z + \ldots + \sigma_{r-1} z^{r-1} + z^r$$

$$\omega(z) = \omega_0 + \omega_1 z + \ldots + \omega_{r-1} z^{r-1}$$

이 된다.

이제 Goppa 코드를 사용하여 오류를 수정할 준비 끝

Encoding and Decoding with Goppa Code

메세지 (0, 0, 1, 1, 0) 을 보내기 위해 인코딩. ([9, ≥ 1, ≥ 3] Goppa Code)

$$(1,0,0,0,1,0,0,1,1) = (0,0,1,1,0)\begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

[9, 5, ≥ 3] 이므로 우린 r ≤ $\frac{3}{2}$ 의 오류를 만들 수 있다. 5번째 자리를 오류로 추가하여

y = (1, 0, 0, 0, 0, 0, 0, 1, 1) 를 전송하면 수신자는 앞의 디코딩 알고리즘을 사용한다.

Encoding & Correcting Errors

Algorithm 3.1 *(Correcting $r \leq \lfloor \frac{t}{2} \rfloor$ Errors in a Goppa Code)*

Let $y = (y_1, \ \ldots \ , \ y_n)$ be a received codeword containing r errors for $2r \leq t$.

1. *Compute the syndrome*

$$s(z) = \sum_{i=1}^{n} \frac{y_i}{z - \alpha_i}.$$

2. *Solve the key equation*

$$\sigma(z)s(z) \equiv \omega(z) \ (mod \ g(z)),$$

by writing

$$\begin{aligned} \sigma(z) &= \sigma_0 + \sigma_1 z + \ldots + \sigma_{r-1} z^{r-1} + z^r, \\ \omega(z) &= \omega_0 + \omega_1 z + \ldots + \omega_{r-1} z^{r-1}, \end{aligned}$$

and solving the accessory system of t equations and $2r$ unknowns. If the code is binary, one can take $\omega(z) = \sigma'(z)$.

3. *Determine the set of error locations $B = \{i \mid \sigma(\alpha_i) = 0\}$.*

4. *Compute the error values $e_i = \frac{\omega(\alpha_i)}{\sigma'(\alpha_i)}$ for all $i \in B$.*

5. *The error vector $e = (e_1, \ \ldots \ , e_n)$ is defined by e_i for $i \in B$ and zeros elsewhere.*

6. *The codeword sent is $c = y - e$.*

Encoding & Correcting Errors

Algorithm 3.1 *(Correcting $r \leq \lfloor \frac{t}{2} \rfloor$ Errors in a Goppa Code)*

Let $y = (y_1, \ldots, y_n)$ be a received codeword containing r errors for $2r \leq t$.

1. *Compute the syndrome*

$$s(z) = \sum_{i=1}^{n} \frac{y_i}{z - \alpha_i}.$$

2. *Solve the key equation*

$$\sigma(z)s(z) \equiv \omega(z) \pmod{g(z)},$$

by writing

$$\begin{aligned} \sigma(z) &= \sigma_0 + \sigma_1 z + \ldots + \sigma_{r-1} z^{r-1} + z^r, \\ \omega(z) &= \omega_0 + \omega_1 z + \ldots + \omega_{r-1} z^{r-1}, \end{aligned}$$

and solving the accessory system of t equations and $2r$ unknowns. If the code is binary, one can take $\omega(z) = \sigma'(z)$.

3. *Determine the set of error locations $B = \{i \mid \sigma(\alpha_i) = 0\}$.*

4. *Compute the error values $e_i = \frac{\omega(\alpha_i)}{\sigma'(\alpha_i)}$ for all $i \in B$.*

5. *The error vector $e = (e_1, \ldots, e_n)$ is defined by e_i for $i \in B$ and zeros elsewhere.*

6. *The codeword sent is $c = y - e$.*

Step 1. * y = (1, 0, 0, 0, 0, 0, 0, 1, 1)

$$\begin{aligned} s(z) &= \sum_{i=1}^{9} \frac{y_i}{z - \alpha_i} \\ &= \frac{1}{z - \alpha} + \frac{1}{z - \alpha^8} + \frac{1}{z - \alpha^9} \\ &\equiv (\alpha^8 + \alpha^4 + \alpha^{10}) + (\alpha^7 + \alpha^{11} + \alpha)z \\ &= 1 + \alpha^{10} z. \end{aligned}$$

$$* H = \begin{pmatrix} \alpha^8 & \alpha & \alpha^5 & \alpha^2 & 1 & \alpha^{10} & \alpha^4 & \alpha^4 & \alpha^{10} \\ \alpha^7 & \alpha^{14} & \alpha^2 & \alpha^{13} & \alpha^{10} & \alpha^4 & \alpha^{12} & \alpha^{11} & \alpha \end{pmatrix}$$

$GF(2^4)$

Therefore, $GF(2^4)^* = < \alpha >$, or equivalently,

$$GF(2^4) = \{0, \alpha, \alpha^2, \ \ldots \ , \alpha^{14}\}.$$

We represent the elements of $GF(2^4)^*$ as the powers of α, using $\alpha^4 = \alpha + 1$. Of course, we represent the element 0 as $(0,0,0,0)^T$.

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

Encoding & Correcting Errors

Algorithm 3.1 *(Correcting $r \le \lfloor \frac{t}{2} \rfloor$ Errors in a Goppa Code)*

Let $y = (y_1, \ldots, y_n)$ be a received codeword containing r errors for $2r \le t$.

1. *Compute the syndrome*

$$s(z) = \sum_{i=1}^{n} \frac{y_i}{z - \alpha_i}.$$

2. *Solve the key equation*

$$\sigma(z)s(z) \equiv \omega(z) \pmod{g(z)},$$

by writing

$$\begin{aligned} \sigma(z) &= \sigma_0 + \sigma_1 z + \ldots + \sigma_{r-1} z^{r-1} + z^r, \\ \omega(z) &= \omega_0 + \omega_1 z + \ldots + \omega_{r-1} z^{r-1}, \end{aligned}$$

and solving the accessory system of t equations and $2r$ unknowns. If the code is binary, one can take $\omega(z) = \sigma'(z)$.

3. *Determine the set of error locations $B = \{i \mid \sigma(\alpha_i) = 0\}$.*
4. *Compute the error values $e_i = \frac{\omega(\alpha_i)}{\sigma'(\alpha_i)}$ for all $i \in B$.*
5. *The error vector $e = (e_1, \ldots, e_n)$ is defined by e_i for $i \in B$ and zeros elsewhere.*
6. *The codeword sent is $c = y - e$.*

Step 2.

* $\sigma(z) = \sigma_0 + \sigma_1 z + \ldots + \sigma_{r-1} z^{r-1} + z^r$

$\omega(z) = \omega_0 + \omega_1 z + \ldots + \omega_{r-1} z^{r-1}$

$\sigma(z)s(z)$ Modulo (z^2 - 1)

$$\begin{aligned} \sigma(z)s(z) &= (\sigma_0 + z)(1 + \alpha^{10} z) \\ &= \sigma_0 + (\alpha^{10}\sigma_0 + 1)z + \alpha^{10} z^2 \\ &\equiv \sigma_0 + (\alpha^{10}\sigma_0 + 1)z + \alpha^{10} \\ &= (\sigma_0 + \alpha^{10}) + (\alpha^{10}\sigma_0 + 1)z, \end{aligned}$$

로부터 다음을 얻는다.

$$\begin{cases} \omega_0 = \sigma_0 + \alpha^{10}, \\ 0 = \alpha^{10}\sigma_0 + 1. \end{cases}$$

그러므로 $\sigma_0 = \alpha^5, \omega_0 = 1$

결론 : $\sigma(z) = z + \alpha^5$, $\omega(z) = 1$

Encoding & Correcting Errors

Algorithm 3.1 *(Correcting $r \leq \lfloor \frac{t}{2} \rfloor$ Errors in a Goppa Code)*

Let $y = (y_1, \ldots, y_n)$ be a received codeword containing r errors for $2r \leq t$.

1. *Compute the syndrome*

$$s(z) = \sum_{i=1}^{n} \frac{y_i}{z - \alpha_i}.$$

2. *Solve the key equation*

$$\sigma(z)s(z) \equiv \omega(z) \pmod{g(z)},$$

by writing

$$\begin{aligned} \sigma(z) &= \sigma_0 + \sigma_1 z + \ldots + \sigma_{r-1} z^{r-1} + z^r, \\ \omega(z) &= \omega_0 + \omega_1 z + \ldots + \omega_{r-1} z^{r-1}, \end{aligned}$$

and solving the accessory system of t equations and $2r$ unknowns. If the code is binary, one can take $\omega(z) = \sigma'(z)$.

3. *Determine the set of error locations $B = \{i \mid \sigma(\alpha_i) = 0\}$.*
4. *Compute the error values $e_i = \frac{\omega(\alpha_i)}{\sigma'(\alpha_i)}$ for all $i \in B$.*
5. *The error vector $e = (e_1, \ldots, e_n)$ is defined by e_i for $i \in B$ and zeros elsewhere.*
6. *The codeword sent is $c = y - e$.*

Step 3

* $\sigma(z) := \prod_{i \in B} (z - \alpha_i)$.

오류 위치 B를 찾는다.

$$\sigma(z) = z + \alpha^5 \ , \ \omega(z) = 1$$

$$B = \{i \mid \sigma(\alpha_i) = 0\} = \{5\}$$

Encoding & Correcting Errors

Algorithm 3.1 *(Correcting $r \leq \lfloor \frac{t}{2} \rfloor$ Errors in a Goppa Code)*

Let $y = (y_1, \ldots, y_n)$ be a received codeword containing r errors for $2r \leq t$.

1. *Compute the syndrome*

$$s(z) = \sum_{i=1}^{n} \frac{y_i}{z - \alpha_i}.$$

2. *Solve the key equation*

$$\sigma(z)s(z) \equiv \omega(z) \pmod{g(z)},$$

by writing

$$\begin{aligned} \sigma(z) &= \sigma_0 + \sigma_1 z + \ldots + \sigma_{r-1} z^{r-1} + z^r, \\ \omega(z) &= \omega_0 + \omega_1 z + \ldots + \omega_{r-1} z^{r-1}, \end{aligned}$$

and solving the accessory system of t equations and $2r$ unknowns. If the code is binary, one can take $\omega(z) = \sigma'(z)$.

3. *Determine the set of error locations $B = \{i \mid \sigma(\alpha_i) = 0\}$.*

4. *Compute the error values $e_i = \frac{\omega(\alpha_i)}{\sigma'(\alpha_i)}$ for all $i \in B$.*

5. *The error vector $e = (e_1, \ldots, e_n)$ is defined by e_i for $i \in B$ and zeros elsewhere.*

6. *The codeword sent is $c = y - e$.*

Step 4.

오류 값은 Binary 이기 때문에 → 1

Encoding & Correcting Errors

Algorithm 3.1 *(Correcting $r \leq \lfloor \frac{t}{2} \rfloor$ Errors in a Goppa Code)*

Let $y = (y_1, \ldots, y_n)$ be a received codeword containing r errors for $2r \leq t$.

1. *Compute the syndrome*

$$s(z) = \sum_{i=1}^{n} \frac{y_i}{z - \alpha_i}.$$

2. *Solve the key equation*

$$\sigma(z)s(z) \equiv \omega(z) \pmod{g(z)},$$

by writing

$$\begin{aligned} \sigma(z) &= \sigma_0 + \sigma_1 z + \ldots + \sigma_{r-1} z^{r-1} + z^r, \\ \omega(z) &= \omega_0 + \omega_1 z + \ldots + \omega_{r-1} z^{r-1}, \end{aligned}$$

and solving the accessory system of t equations and $2r$ unknowns. If the code is binary, one can take $\omega(z) = \sigma'(z)$.

3. *Determine the set of error locations $B = \{i \mid \sigma(\alpha_i) = 0\}$.*

4. *Compute the error values $e_i = \frac{\omega(\alpha_i)}{\sigma'(\alpha_i)}$ for all $i \in B$.*

5. *The error vector $e = (e_1, \ldots, e_n)$ is defined by e_i for $i \in B$ and zeros elsewhere.*

6. *The codeword sent is $c = y - e$.*

Step 5.

$$B = \{i \mid \sigma(\alpha_i) = 0\} = \{5\}$$

오류벡터 e = (0, 0, 0, 0, 1, 0, 0, 0, 0)

Encoding & Correcting Errors

Algorithm 3.1 *(Correcting $r \leq \lfloor \frac{t}{2} \rfloor$ Errors in a Goppa Code)*

Let $y = (y_1, \ldots, y_n)$ be a received codeword containing r errors for $2r \leq t$.

1. *Compute the syndrome*

$$s(z) = \sum_{i=1}^{n} \frac{y_i}{z - \alpha_i}.$$

2. *Solve the key equation*

$$\sigma(z)s(z) \equiv \omega(z) \pmod{g(z)},$$

by writing

$$\begin{aligned} \sigma(z) &= \sigma_0 + \sigma_1 z + \ldots + \sigma_{r-1} z^{r-1} + z^r, \\ \omega(z) &= \omega_0 + \omega_1 z + \ldots + \omega_{r-1} z^{r-1}, \end{aligned}$$

and solving the accessory system of t equations and $2r$ unknowns. If the code is binary, one can take $\omega(z) = \sigma'(z)$.

3. *Determine the set of error locations $B = \{i \mid \sigma(\alpha_i) = 0\}$.*
4. *Compute the error values $e_i = \frac{\omega(\alpha_i)}{\sigma'(\alpha_i)}$ for all $i \in B$.*
5. *The error vector $e = (e_1, \ldots, e_n)$ is defined by e_i for $i \in B$ and zeros elsewhere.*
6. *The codeword sent is $c = y - e$.*

Step 6

y = (1, 0, 0, 0, 0, 0, 0, 1, 1)

오류벡터 e = (0, 0, 0, 0, 1, 0, 0, 0, 0)

c = y - e = (1, 0, 0, 0, 1, 0, 0, 1, 1)

Decoding

오류수정을 통하여 수신한 *y*로부터 올바른 코드워드 *c* 를 찾았고, *G* 는 알고있기 때문에

다음 식을 통하여 메세지 *m* 을 쉽게 획득할 수 있다.

$$mG = c$$

$$(1,0,0,0,1,0,0,1,1) = (0,0,1,1,0) \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Information Set Decoding

- 코드기반 암호에서 가장 효율적이라고 알려진 공격법
  - 주어진 암호문 $\boldsymbol{c}$ 와 공개키 $\boldsymbol{G'}$ 을 이용하여 원본메세지 $\boldsymbol{m}$ 을 복구함

1. n – bit 의 길이 암호문 $\boldsymbol{c}$ 에서 k – bit 의 벡터 $\boldsymbol{c_k}$ 를 랜덤하게 선택
   → 오류 위치를 모르는 상태에서 자신이 선택한 k – bit 벡터에 오류가 포함되지 않아야 함

2. 선택한 열의 index에 맞춰 $\boldsymbol{G'}$ 으로부터 $\boldsymbol{G'_k}$ 를 뽑아낸다. 이때, $\boldsymbol{G'_k}$ 는 invertible

3. $\boldsymbol{c} + \boldsymbol{c_k}\boldsymbol{G'_k}^{-1}\boldsymbol{G}$ 의 weight가 t 와 같거나 더 적은지 확인한다. 그렇지 않으면 1단계 부터 다시 반복

4. 앞의 조건이 만족한다면 원본메세지 $\boldsymbol{m} = \boldsymbol{c_k}\boldsymbol{G'_k}^{-1}$ 로 복구가 가능하다.

Information Set Decoding Example

1. n – bit 의 길이 암호문 $\boldsymbol{c}$ 에서 k – bit 의 벡터 $\boldsymbol{c_k}$ 를 랜덤하게 선택
   → 오류 위치를 모르는 상태에서 자신이 선택한 k – bit 벡터에 오류가 포함되지 않아야 함

$$G' = S\,G\,P = \begin{matrix} 1\ 1\ 1\ 1\ 0\ 0\ 0 \\ 1\ 1\ 0\ 0\ 1\ 0\ 0 \\ 1\ 0\ 0\ 1\ 1\ 0\ 1 \\ 0\ 1\ 0\ 1\ 1\ 1\ 0 \end{matrix} \qquad m = 1\ 1\ 0\ 1$$

$$e = 0\ 0\ 0\ 0\ 1\ 0\ 0$$

$$c = 0\ 1\ 1\ 0\ 1\ 1\ 0$$

$$c_k = 0\ 1\ 1\ 1 G'_k = \begin{matrix} 1\ 1\ 1\ 0 \\ 1\ 1\ 0\ 0 \\ 1\ 0\ 0\ 0 \\ 0\ 1\ 0\ 1 \end{matrix}$$

$$G'_k{}^{-1} = \begin{matrix} 0\ 0\ 1\ 0 \\ 0\ 1\ 1\ 0 \\ 1\ 1\ 0\ 0 \\ 0\ 1\ 1\ 1 \end{matrix}$$

밑의 결과 벡터의 weight를 확인한다. (t or less)

$$c + c_k G'_k{}^{-1} G = 0\ 1\ 1\ 0\ 1\ 1\ 0 + 0\ 1\ 1\ 0\ 0\ 1\ 0 = 0\ 0\ 0\ 0\ 1\ 0\ 0$$

마지막으로 최종 메세지 m = $c_k G'_k{}^{-1}$ = 1 1 0 1

$n = 1024, k = 524$ 의 $Goppa$ 코드에 대하여 이 경우의 수는 $2^{80.7}$ 결과의 작업 계수

NIST Round 2 Code-based PQC

NIST Round 2

- 양자 시대에 대비하기 위한 양자 내성 암호 후보군 모집
- Round 진행: 64개 → 26개
- 현재 남아있는 후보군들은 NIST의 보안 테스트 합격
- 성능에 대한 최적화가 요구되고 있음
- 코드 기반 암호 7개

  ClassicMcEliece, BIKE, RQC, HQC, LEDAcrypt, ROLLO, NTS-KEM

NIST Round 2 - Classic McEliece

- 최초의 코드기반암호 McEliece의 업그레이드 버전
- 초기의 McEliece는 OW-CPA(One-Way Chosen-Plaintext Attack)로 설계
  - 코드워드가 랜덤하게 선택되었을 때, 공격자가 암호문과 공개키로부터 코드워드를 찾아내기 어려움
- One-Way 시스템은 공격자가 임의로 수정한 암호텍스트를 보내어 수신자의 반응을 확인하는 공격이 있을 수 있음
  - 기존 McEliece PKE는 그러한 공격에 내성이 없었음
- 기존 OW-CPA의 PKE에서 ROM(Random Oracle Model) attack 에 대항하여 IND-CCA2를 보장하는 KEM(Key-Encapsulation Mechanism)으로 효율적으로 변환하며 보안성을 강화함
- QROM(Quantum Random Oracle Model) Attack 에 대해서도 견고성을 획득
- 최종적으로 Classic McEliece는 IND-CCA2를 위한 양자 컴퓨터상에서도 안전한 높은 보안레벨의 KEM 디자인을 제안
- 양자 공개키 암호화의 장기보안에 확신을 가질수 있도록 설계되었다 주장
- 핵심강점은 보안
  - 기존의 Goppa 코드 대신 다른 코드를 사용함으로써 효율성을 늘리는 것보다 오랜기간 안전성을 지속한 Goppa 코드 사용을 고수
- 암호문의 경우 양자내성암호중에 매우 작은 편

NIST Round 2 - BIKE

- BIKE의 모든 버전에서는 Bit-Flipping 디코더를 통해 복호를 효율적으로 수행하는 QC-MDPC 코드를 사용
  - Bit-Flipping 디코더는 가장 틀린 위치가 무엇인지 추정하고, 뒤집어서 그 결과가 이전보다 좋았는지 syndrome weight 관찰

- Round 1에서는 비교적 높은 복호 실패율을 보여줬음
  - IND-CCA같은 더 높은 보안개념을 달성하지 못하게 하였다.

- Round 2에서는 향상된 Backflip 디코더라는 향상된 디코딩 기술을 적용함으로써 무시가능한 수준의 복호 실패율을 달성함

- IND-CCA 보안성 버전인 BIKE-1-CCA, BIKE-2-CCA, BIKE-3-CCA를 구현
  - 약간의 더 높은 비용을 요구하는 대신 정적 키를 사용함으로 써 더 높은 보안 레벨을 제공한다.

NIST Round 2 - NTS-KEM

- NTS-KEM은 McEliece와 Niederreiter의 변형
  - 코드의 길이가 증가해도 암호문의 길이는 매우 미세하게 증가
  - 하드웨어나 소프트웨어의 구현 상에서 재 설계 없이 매개변수의 조정만으로 보안수준을 결정 가능
- 암호문이 비교적 작고, 밴드폭이 어플리케이션 상의 한정된 밴드폭 통신상에서 적합
- 큰 공개키 크기를 요구
- 40년동안 지켜진 Goppa 코드를 사용한 보수적인 디자인을 통한 강한 보안 수준 보장
- NTS-KEM도 BIKE와 같이 보안레벨 1, 3, 5수준의 3가지 버전을 제안
  - 매개변수 설정에서 높은 유연성을 제공
- ROM(random oracle model)에 대한 보안성 확보
  - QROM 에 대한 대비 필요

NIST Round 2 - HQC

- HQC는 Quasi-Cyclic 코드를 사용
- IND-CCA2 보안성을 제공
- KEM변환에 완벽히 부합하며 보안레벨 1, 3, 5에 따른 세가지 버전을 제공
- 복호 시 실패에 대해 DFR(Decryption failure Rate) 분석을 특징으로한다.
- QC 코드는 암호체계에서 매우 유용하며 상당히 큰 키의 사이즈를 줄임
- 느린 암호화 속도

NIST Round 2 - RQC

- RQC는 Rank-Quasi Cyclic 코드를 기반
- IND-CCA2 의 보안성을 제공
- KEM 변환에 부합하며 보안강도에 따라 1, 3, 5 세가지 버전을 제공
- RQC의 디코딩알고리즘은 deterministic 하기 때문에 Decryption Failure Rate(DFR)이 제로
- 부채널 공격에 대한 내성 확보

NIST Round 2 - ROLLO

- Round1 의 암호후보 3개Rank-Ouroboros, LAKE, LOCKER 를 통합한 버전
- LAKE는 ROLLO-I, LOCKER는 ROLLO-II, Rank Ouroboros는 ROLLO-III 로 개명
- Rank Metric 코드 기반으로하며 LRPC(Low Rank Parity Check)코드에 대하여 동일한 디코딩 알고리즘을 공유함
- 부채널 공격을 막기위해 ROLLO-I, III 는 임시키를 사용하여 공격을 무효화 시키고 ROLLO-II의 경우 키에대한 정보를 얻을 확률이 낮아 공격이 실용적이지 않다고 주장한다.
- ROLLO-I, ROLLO-III는 IND-CPA KEM 보장
- ROLLO-II는 IND-CCA2 PKE 이보장

NIST Round 2 - LEDAcrypt

- LEDAcrypt는 크게 다음 3가지로 분류

1. 임시키를 사용하는 LEDAcrypt KEM 은 IND-CPA 보안을 제공

2. long-term key를 사용하는 새로운 버전의 LEDAcrypt KEM은 DEM(Data Encapsulation mechanism)과 함께 IND-CCA2 보안을 제공

3. LEDAcrypt KEM 의 대체 솔루션으로 long-term key를 사용하는 LEDAcrypt PKC는 짧은 메세지를 사용하는 대역폭에 최적화 되어있으며 IND-CCA2 보안을 제공

- 보안레벨 1, 3, 5에 따른 버전을 제공하며 QC-LDPC(Quasi-Cyclic Low-Density Parity- Check) 코드에 기반함

NIST Round 2 - Classic McEliece

Classic McEliece 성능 비교

	Intel NUC			Raspberry Pii		
	KeyGen	Encaps	Decaps	KeyGen	Encaps	Decaps
mceliece348864	164.82	0.0696	23.88	1780.94	0.84	247.94
mceliece460896	339.86	0.15	56.6	3864.43	2.52	630.32
mceliece6688128	939.42	0.20	108.20	8987.07	2.74	9893.79
mceliece6960119	638.54	0.47	103.28	7003.93	6.64	1138.87
mceliece8192128	534.11	0.23	140.84	5738.65	3.05	1709.16

NIST Round 2 - BIKE

BIKE 성능 비교

	Intel NUC			Raspberry Pi		
	KeyGen	Encaps	Decaps	KeyGen	Encaps	Decaps
BIKE-1 CPA(LEVEL1)	0.26	0.22	1.49	2.68	2.69	11.8
BIKE-1 CPA(LEVEL3)	0.66	0.57	3.99	7.10	6.99	2.94
BIKE-1 CPA(LEVEL5)	1.00	0.89	8.97	10.98	10.89	73.8
BIKE-2 CPA(LEVEL1)	1.74	0.129	1.41	30.79	1.62	10.83
BIKE-2 CPA(LEVEL3)	6.69	0.31	3.75	80.86	3.72	23.93
BIKE-2 CPA(LEVEL5)	11.41	0.48	8.56	131.81	6.26	66.51
BIKE-3 CPA(LEVEL1)	0.23	0.27	1.83	1.51	2.95	12.24
BIKE-3 CPA(LEVEL3)	0.41	0.64	4.17	4.07	8.04	31.37
BIKE-3 CPA(LEVEL5)	0.86	1.30	99.03	8.51	16.74	79.88
BIKE-1 CCA(LEVEL1)	0.37	0.28	1.85	3.40	3.46	19.14
BIKE-1 CCA(LEVEL3)	0.89	0.73	4.40	9.08	9.70	49.17
BIKE-1 CCA(LEVEL5)	1.74	1.60	9.45	20.53	20.97	113.41
BIKE-2 CCA(LEVEL1)	2.28	0.16	1.61	37.86	1.86	16.11

NIST Round 2 - BIKE

BIKE 성능 비교

	Intel NUC			Raspberry Pi		
	KeyGen	Encaps	Decaps	KeyGen	Encaps	Decaps
BIKE-2 CCA(LEVEL3)	8.96	0.38	3.72	105.13	4.79	38.95
BIKE-2 CCA(LEVEL5)	18.62	0.83	8.02	231.88	10.79	93.15
BIKE-3 CCA(LEVEL1)	0.24	0.29	1.95	1.88	3.81	18.26
BIKE-3 CCA(LEVEL3)	0.62	8.09	4.51	5.21	10.66	50.94
BIKE-3 CCA(LEVEL5)	1.12	1.87	10.21	12.53	24.98	122.57

NIST Round 2 - NTS-KEM

NTS-KEM 성능 비교

	Intel NUC			Raspberry Pi		
	KeyGen	Encap	Decap	KeyGen	Encap	Decap
nts_kem_12_64	55.96	0.12	1.71	291.66	1.28	9.8
nts_kem_13_80	159.37	0.28	3.39	958.46	2.99	17.80
nts_kem_13_136	277.88	0.37	6.80	2513.81	3.892	35.57

NIST Round 2 - HQC

HQC 성능 비교

	Intel NUC			Raspberry Pi		
	KeyGen	Encap	Decap	KeyGen	Encap	Decap
hqc-128-1	3.96	0.76	1.24	50.65	9.02	15.30
hqc-192-1	0.93	1.83	2.82	11.06	24.12	37.09
hqc-192-2	1.01	1.94	2.93	12.68	25.94	39.97
hqc-256-1	1.30	2.56	3.92	16.64	35.57	51.61
hqc-256-2	1.59	3.11	4.73	17.85	34.61	52.12
hqc-256-3	1.80	3.60	5.46	19.13	40.	60.05

NIST Round 2 - RQC

RQC 성능 비교

	Intel NUC			Raspberry Pi		
	KeyGen	Encaps	Decaps	KeyGen	Encaps	Decaps
rqc128	0.31	0.55	2.61	2.64	4.65	27.03
rqc192	0.46	0.91	5.75	4.19	8.50	64.91
rqc256	0.86	1.75	11.99	7.67	16.98	126.34

NIST Round 2 - ROLLO

ROLLO 성능 비교

	Intel NUC			Raspberry Pi		
	KeyGen	Encaps	Decaps	KeyGen	Encaps	Decaps
Rollo-I-128	0.84	0.17	0.53	8.19	1.21	4.27
Rollo-I-192	1.62	0.21	0.99	10.18	1.48	6.96
Rollo-I-256	1.70	0.31	1.55	18.94	2.48	12.46
Rollo-II-128	7.16	0.82	2.23	73.94	6.89	19.84
Rollo-II-192	7.70	0.92	2.62	89.42	7.97	24.07
Rollo-II-256	7.77	1.03	2.91	93.77	8.69	27.02
Rollo-III-128	0.16	0.30	0.49	1.67	2.62	3.98
Rollo-III-192	0.20	0.41	0.96	2.19	3.56	7.53
Rollo-III-256	0.37	0.73	1.65	3.46	5.94	12.66

NIST Round 2 - LEDAcrypt

LEDAcrypt 성능 비교

	Intel NUC			Raspberry Pi		
	KeyGen	Encap	Decap	KeyGen	Encap	Decap
LEDAcrypt ephemeral keys 1-2	8.66	0.737	3.041	79.62	7.89	29.91
LEDAcrypt ephemeral keys 1-3	2.641	0.548	3.301	28.16	5.71	34.09
LEDAcrypt ephemeral keys 1-4	2.599	0.706	5.250	27.95	7.63	53.12
LEDAcrypt ephemeral keys 3-2	21.237	1.344	7.945	208.15	12.94	80.64
LEDAcrypt ephemeral keys 3-3	8.801	1.429	8.078.	86.61	15.07	81.22
LEDAcrypt ephemeral keys 3-4	7.907	1.863	12.023	81.02	17.13	122.37
LEDAcrypt ephemeral keys 5-2	43.430	2.675	13.400	417.12	29.05	134.20
LEDAcrypt ephemeral keys 5-3	25.234	2.778	17.188	256.30	29.97	173.21
LEDAcrypt ephemeral keys 5-4	18.651	3.350	17.561	180.91	36.33	179.01
LEDAcrypt Longterm keys	390.139	2.811	3.466	3911.10	30.22	36.07

NIST Round 2 - LEDAcrypt

LEDAcrypt 성능 비교

LEDAcrypt Longterm keys 1-2(2 ^-128)	581.784	3.905	3.746	5722.01	41.01	38.91
LEDAcrypt Longterm keys 3-2(2 ^-64)	926.896	4.602	8.505	9273.56	45.12	85.15
LEDAcrypt Longterm keys 3-2(2 ^-192)	2135.948	9.984	9.274	20059.0	101.16	95.65
LEDAcrypt Longterm keys 5-2(2 ^-64)	3665.445	7.818	14.634	38056.9	80.12	149.94
LEDAcrypt Longterm keys 5-2(2 ^-256)	4467.717	18.410	17.787	42680.2	186.55	178.07
LEDAcrypt PKC 1-2(2^-64)	302.381	2.885	3.283	3126.65	29.01	33.73
LEDAcrypt PKC 1-2(2^-128)	432.130	4.125	4.303	4123.01	43.21	44.81
LEDAcrypt PKC 3-2(2^-64)	926.896	4.602	8.505	8971.91	48.72	87.15
LEDAcrypt PKC 3-2(2^-192)	2135.948	9.984	9.274	2317.43	102.31	93.01
LEDAcrypt PKC 5-2(2^-64)	4415.511	7.954	16.393	42458.7	81.22	162.53
LEDAcrypt PKC 5-2(2^-256)	4555.365	18.819	16.856	45355.2	189.27	170.51

NIST Round 2

	Code	Security history	Structure	Security level	Performance
Classic McEliece	Goppa	long	KEM	IND-CCA2	low
BIKE	QC-MDPC	short	KEM	IND-CCA	high
NTS-KEM	Goppa	long	KEM	IND-CCA	low
HQC	QC	short	KEM	IND-CCA2	high
RQC	QC	short	KEM	IND-CCA2	high
ROLLO	Rank Metric	short	PKE/KEM	IND-CPA(KEM) IND-CCA2(PKE)	high
LEDAcrypt	QC-LDPC	short	PKE/KEM	IND-CCA2	low

Q & A

감사합니다