Grover on ClassicMcEliece

장경배

https://youtu.be/q7Vs4nJ3q2Y

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB

Remind

Code-based Cryptography

- **같은** Goppa code(G, H)를 사용하는 경우

> **4.1 Equivalence with McEliece PKC**
>
> As mentioned earlier in the text that original Niederreiter scheme worked for Generalized Reed Solomon codes. This lead to a major security threat as mentioned by Sidelnikov and Shestakov [26]. Though it was earlier stated that the scheme of Niederreiter and McEliece worked alike provided they use same underlying code i.e., binary Goppa code. The benefit of using Niederreiter scheme was that the size of key needed was much reduced. The question of showing both these schemes can be converted into each other, or their equivalence relation is described as below.

- **Niederreiter System**의 문제를 푸는 것은 **McEliece System**을 푸는 것과 동일

$$C_0 = He \quad \cong \quad C = mG' + e$$

Parity check Matrix $H \rightarrow (n-k) \times n$ Generator Matrix $G' \rightarrow (k \times n)$

- **McEliece System** to **Niederreiter System**

$$C(H)^T = \cancel{mG'(H)^T} + e(H)^T$$

$$C(H)^T = \boxed{e(H)^T}$$

Information Set Decoding (ISD)

- **Prange's ISD**

1. n –bit 의 길이 암호문 $\boldsymbol{C}$ 에서 k–bit 의 벡터 $\boldsymbol{C_k}$ 를 랜덤하게 선택

   → 오류 위치를 모르는 상태에서 공격자는 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. $C_k {G'_k}^{-1} = (mG' + e)_k {G'_k}^{-1} = m + e_k {G'_k}^{-1} \rightarrow C + C_k {G'_k}^{-1} G$의 weight가 t인지 확인

   $(\boldsymbol{mG} + \boldsymbol{e}) + \boldsymbol{mG} = \boldsymbol{e}$

4. 앞의 조건이 만족한다면 원본 메시지 ($\boldsymbol{m} = \boldsymbol{C_k} {\boldsymbol{G'_k}}^{-1}$) 로 복구 가능

$$\underbrace{\begin{bmatrix} m \end{bmatrix}}_{1 \times 2720} \times \underbrace{\begin{bmatrix} G \end{bmatrix}}_{2720 \times 3488} + \underbrace{\begin{bmatrix} 1 & 1 & e & 1 \end{bmatrix}}_{1 \times 3488} = \underbrace{\begin{bmatrix} C \end{bmatrix}}_{1 \times 3488}$$

Information Set Decoding (Toy)

1. n–bit 의 길이 암호문 (c)에서 k–bit의 벡터 (c_k)를 무작위 선택

$$G' = S\,G\,P = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Secret

$m = 1\ 1\ 0\ 1$

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

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

$c_k = 0\ 1\ 1\ 1$

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

2. 이때, G'_k 는 invertible, $G'_k{}^{-1}$ 계산 $\rightarrow$ $G'_k{}^{-1} = \begin{matrix} 0\,0\,1\,0 \\ 0\,1\,1\,0 \\ 1\,1\,0\,0 \\ 0\,1\,1\,1 \end{matrix}$

3. 밑의 결과 벡터의 weight를 확인 (t)

$c + c_k G'_k{}^{-1} G = 0\ 1\ 1\ 0\ 1\ 1\ 0 + 0\ 1\ 1\ 0\ 0\ 1\ 0 = 0\ 0\ 0\ 0\ 1\ 0\ 0$, 최종 메시지 $m = c_k G'_k{}^{-1} = 1\ 1\ 0\ 1$

ISD on ClassicMcEliece

Information Set Decoding on CM

- **Parity check 행렬을 공개키로 사용하는 Niederreiter system의 경우 ISD 접근방식이 조금 다름**
  - **공격자가 선택한 Columns에 오류가 모두 포함되어 있어야 함**

(*iv*) The matrix $\hat{H}$ becomes

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

(*v*) On applying Gaussian elimination, the matrix H becomes

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

$(n-k) \times n$, (8 ×16)

$\boldsymbol{e} = (1100000000000000), (n = 16, t = 2)$

Secret

$\boldsymbol{C} = \boldsymbol{He} = (11000000)$

Information Set Decoding on CM

1. 오류 위치를 모르는 공격자는 랜덤하게 $(n-k)$ Columns 선택

   – **선택한 $(n-k)$ Columns에 모두 오류가 포함되어 있어야함**

2. 선택한 $(n-k)$ Columns에 따라 H 재구성 $\rightarrow$ H_{n-k}

   – **재구성한 행렬 H_{n-k}가 Invertible 해야함**

3. H_{n-k} 에 Gaussian Elimination을 수행, H_{n-k}의 역행렬 $(H_{n-k})^{-1}$ 계산

4. **$(H_{n-k})^{-1} \times C^T$ 의 Hamming weight가 t 인 경우, 공격 성공**

5. 공격자가 선택한 $(n-k)$ Columns 에 모든 오류가 포함되어 있고, $(H_{n-k})^{-1} \times C^T$는 오류 위치를 알려줌

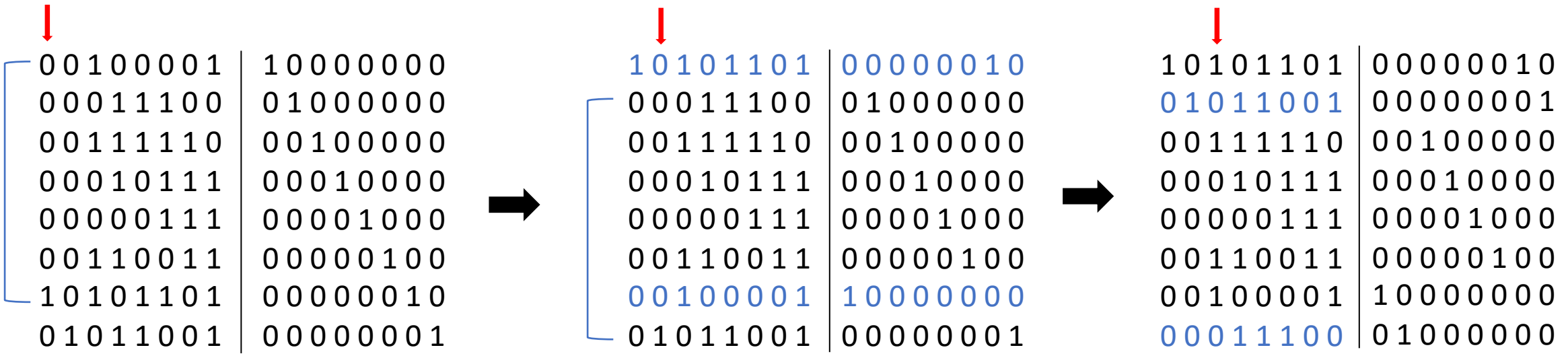
$$H = \left(\begin{array}{cccccc|cc|cccccc|cc} 1&0&0&0&0&0&0&0&1&0&0&0&0&1&1&1\\ 0&1&0&0&0&0&0&0&0&1&1&1&0&0&0&1\\ 0&0&1&0&0&0&0&0&1&1&1&1&1&0&1&1\\ 0&0&0&1&0&0&0&0&0&1&0&1&1&1&0&1\\ 0&0&0&0&1&0&0&0&0&0&0&1&1&1&1&0\\ 0&0&0&0&0&1&0&0&1&1&0&0&1&1&1&0\\ 0&0&0&0&0&0&1&0&1&0&1&1&0&1&0&0\\ 0&0&0&0&0&0&0&1&0&1&1&0&0&1&1&0 \end{array}\right)$$

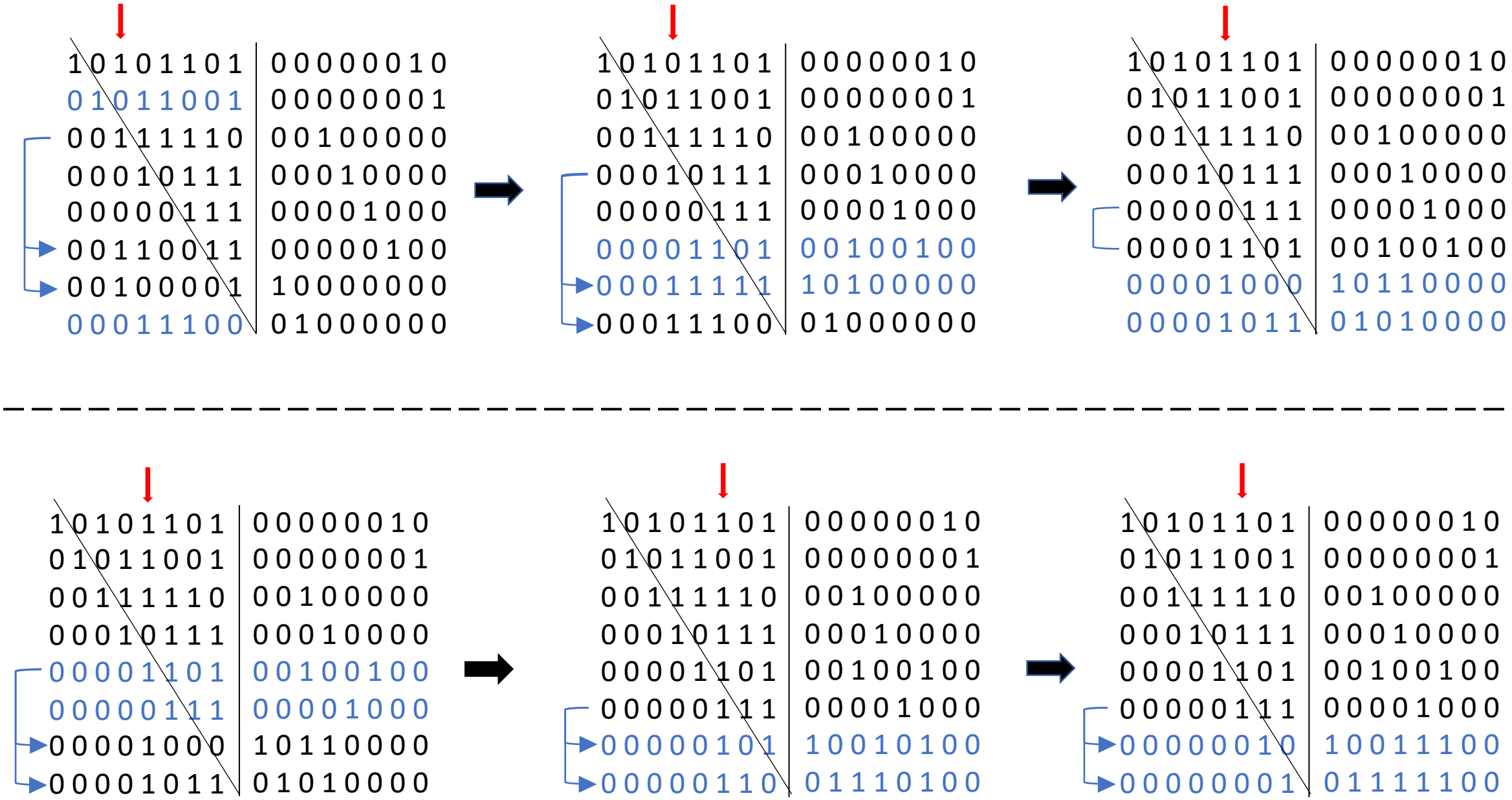
H_{n-k}

$\boldsymbol{e} = (0000001100000000), (n = 16, t = 2)$

Secret

$\boldsymbol{C} = \boldsymbol{He} = (00000011)$




10101101 00000010
01011001 00000001
00111110 00100000
00010111 00010000
00000111 00001000
00110011 00000100
00100001 10000000
00011100 01000000
10101101 00000010
01011001 00000001
00111110 00100000
00010111 00010000
00000111 00001000
00001101 00100100
00011111 10100000
00011100 01000000
10101101 00000010
01011001 00000001
00111110 00100000
00010111 00010000
00000111 00001000
00001101 00100100
00001000 10110000
00001011 01010000
10101101 00000010
01011001 00000001
00111110 00100000
00010111 00010000
00001101 00100100
00000111 00001000
00001000 10110000
00001011 01010000
10101101 00000010
01011001 00000001
00111110 00100000
00010111 00010000
00001101 00100100
00000111 00001000
00000101 10010100
00000110 01110100
10101101 00000010
01011001 00000001
00111110 00100000
00010111 00010000
00001101 00100100
00000111 00001000
00000010 10011100
00000001 01111100


10101101 00000010
01011001 00000001
00111110 00100000
00010111 00010000
00001101 00100100
00000011 00001000
00000010 10011100
00000001 01111100
10101100 01111110
01011000 01111101
00111110 00100000
00010110 01101100
00001100 01011000
00000110 01110100
00000010 10011100
00000001 01111100
10101100 01111110
01011000 01111101
00111100 10111100
00010100 11110000
00001100 01011000
00000100 11101000
00000010 10011100
00000001 01111100
10101000 10010110
01011000 01111101
00111000 01010100
00010000 00011000
00001000 10110000
00000100 11101000
00000010 10011100
00000001 01111100
10100000 00100110
01010000 11001101
00110000 11100100
00010000 00011000
00001000 10110000
00000100 11101000
00000010 10011100
00000001 01111100
10100000 00100110
01000000 11010101
00100000 11111100
00010000 00011000
00001000 10110000
00000100 11101000
00000010 10011100
00000001 01111100

1 0 0 0 0 0 0 0	1 1 0 1 1 0 1 0
0 1 0 0 0 0 0 0	1 1 0 1 0 1 0 1
0 0 1 0 0 0 0 0	1 1 1 1 1 1 0 0
0 0 0 1 0 0 0 0	0 0 0 1 1 0 0 0
0 0 0 0 1 0 0 0	1 0 1 1 0 0 0 0
0 0 0 0 0 1 0 0	1 1 1 0 1 0 0 0
0 0 0 0 0 0 1 0	1 0 0 1 1 1 0 0
0 0 0 0 0 0 0 1	0 1 1 1 1 1 0 0

$$(H_{n-k})^{-1} = \begin{pmatrix} 1\,1\,0\,1\,1\,0\,1\,0 \\ 1\,1\,0\,1\,0\,1\,0\,1 \\ 1\,1\,1\,1\,1\,1\,0\,0 \\ 0\,0\,0\,1\,1\,0\,0\,0 \\ 1\,0\,1\,1\,0\,0\,0\,0 \\ 1\,1\,1\,0\,1\,0\,0\,0 \\ 1\,0\,0\,1\,1\,1\,0\,0 \\ 0\,1\,1\,1\,1\,1\,0\,0 \end{pmatrix}$$

Invertible하지 않다?

만들 수 없음 →

1
0 1
0 0 1
0 0 0 1
0 0 0 0 1
0 0 0 0 0 1
0 0 0 0 0 0 1
0 0 0 0 0 0 0 1

- $(H_{n-k})^{-1} \times C^T$ 의 Hamming weight가 t 인 경우, 공격 성공

$$\begin{pmatrix} 1\,1\,0\,1\,1\,0\,1\,0 \\ 1\,1\,0\,1\,0\,1\,0\,1 \\ 1\,1\,1\,1\,1\,1\,0\,0 \\ 0\,0\,0\,1\,1\,0\,0\,0 \\ 1\,0\,1\,1\,0\,0\,0\,0 \\ 1\,1\,1\,0\,1\,0\,0\,0 \\ 1\,0\,0\,1\,1\,1\,0\,0 \\ 0\,1\,1\,1\,1\,1\,0\,0 \end{pmatrix} \times \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \rightarrow \text{Weight} = 2(t)$$

$(H_{n-k})^{-1}$ C^T

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

$(H_{n-k})^{-1}$ C^T Weight = $2(t)$

- 공격자가 선택한 $(n-k)$ Columns 에 모든 오류가 포함되어 있고, $(H_{n-k})^{-1} \times C^T$는 오류 위치를 알려줌

- 선택한 Columns의 1, 2번째에 bit가 1, 나머지가 모두 0인 벡터가 Secret

$$H = \begin{array}{c} \begin{matrix} 0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0 \end{matrix} \\ \left(\begin{array}{cccccc|cc|cccccccc} 1&0&0&0&0&0&0&0&1&0&0&0&0&1&1&1 \\ 0&1&0&0&0&0&0&0&0&1&1&1&0&0&0&1 \\ 0&0&1&0&0&0&0&0&1&1&1&1&1&0&1&1 \\ 0&0&0&1&0&0&0&0&0&1&0&1&1&1&0&1 \\ 0&0&0&0&1&0&0&0&0&0&0&1&1&1&1&0 \\ 0&0&0&0&0&1&0&0&1&1&0&0&1&1&1&0 \\ 0&0&0&0&0&0&1&0&1&0&1&1&0&1&0&0 \\ 0&0&0&0&0&0&0&1&0&1&1&0&0&1&1&0 \end{array}\right) \end{array}$$

H_{n-k}

$$\boldsymbol{e} = (0000001100000000), (n = 16, t = 2)$$

Secret

$$\boldsymbol{C} = \boldsymbol{He} = (00000011)$$

$$H = \left(\begin{array}{cccccc|cc|cccccc} 1&0&0&0&0&0&0&0&1&0&0&0&0&1&1&1 \\ 0&1&0&0&0&0&0&0&0&1&1&1&0&0&0&1 \\ 0&0&1&0&0&0&0&0&1&1&1&1&1&0&1&1 \\ 0&0&0&1&0&0&0&0&0&1&0&1&1&1&0&1 \\ 0&0&0&0&1&0&0&0&0&0&0&1&1&1&1&0 \\ 0&0&0&0&0&1&0&0&1&1&0&0&1&1&1&0 \\ 0&0&0&0&0&0&1&0&1&0&1&1&0&1&0&0 \\ 0&0&0&0&0&0&0&1&0&1&1&0&0&1&1&0 \end{array}\right)$$

H_{n-k}

$$\boldsymbol{e} = (0000000011000000), (n = 16, t = 2)$$

Secret

$$\boldsymbol{C} = \boldsymbol{He} = (11010011)$$

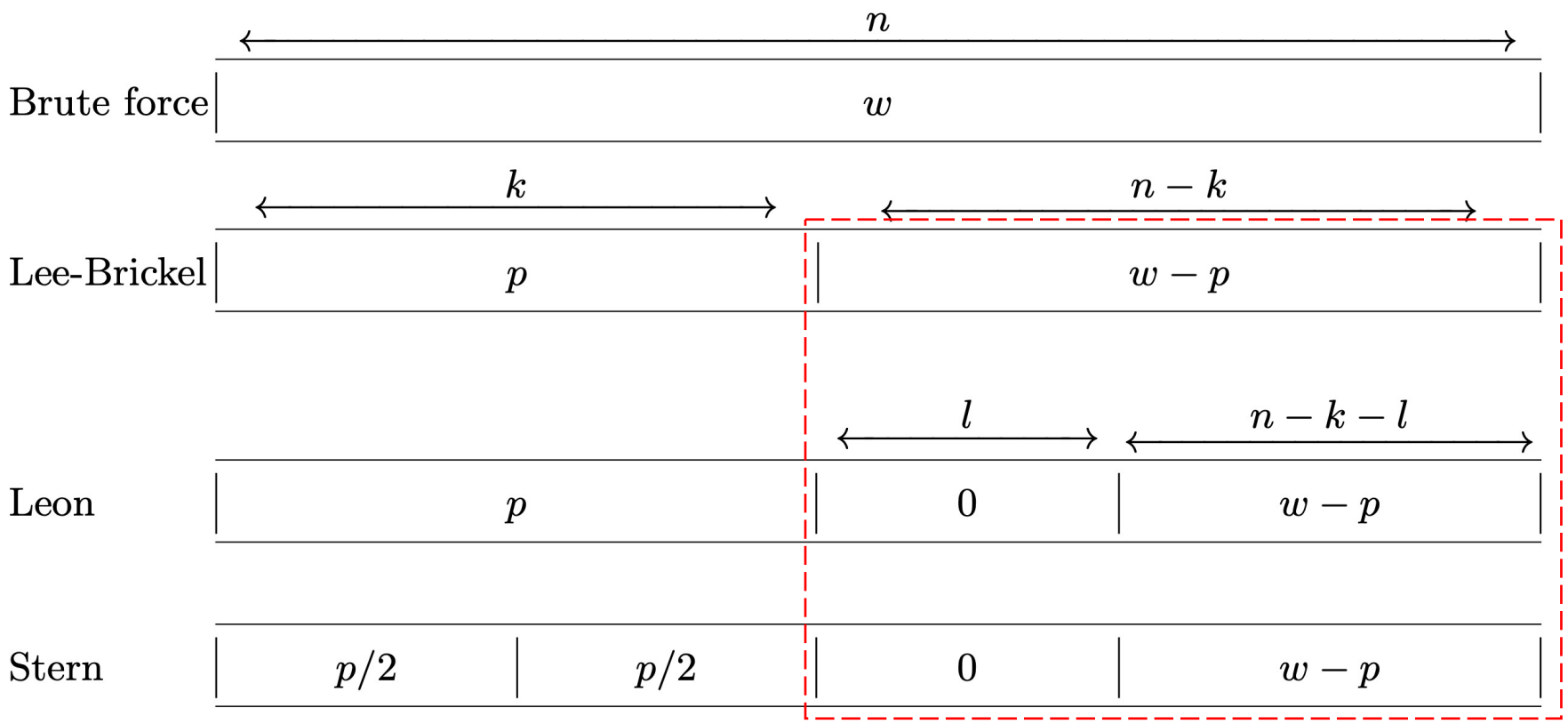
$$\left(\begin{array}{cccccccc} 1&1&0&1&1&0&1&0 \\ 1&1&0&1&0&1&0&1 \\ 1&1&1&1&1&1&0&0 \\ 0&0&0&1&1&0&0&0 \\ 1&0&1&1&0&0&0&0 \\ 1&1&1&0&1&0&0&0 \\ 1&0&0&1&1&1&0&0 \\ 0&1&1&1&1&1&0&0 \end{array}\right) \times \left(\begin{array}{c} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{array}\right) = \left(\begin{array}{c} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}\right)$$

$(H_{n-k})^{-1}$ $\quad$ C^T $\quad$ Weight = $2(t)$

- **선택한 Columns의 2, 3번째에 bit가 1, 나머지가 모두 0인 벡터가 Secret** → (0000000011000000)

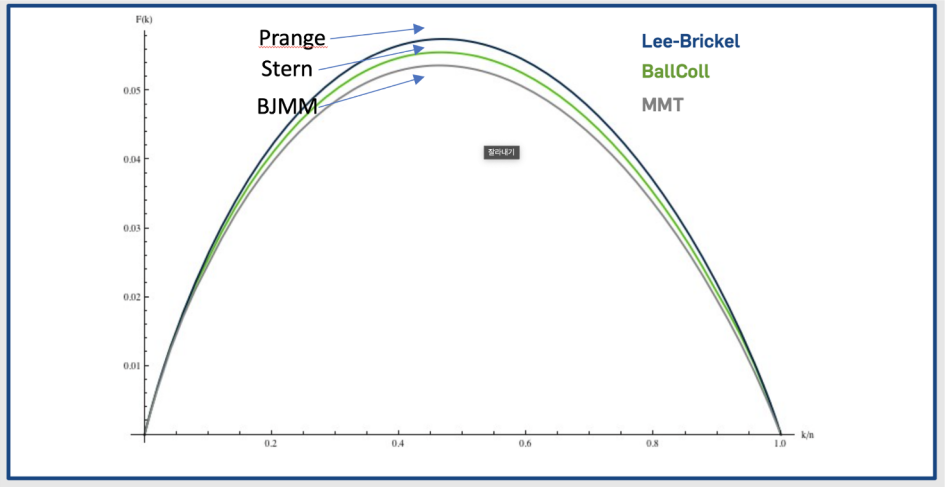
Information Set Decoding (ISD) Variants

- **Prange ISD는 k에서 $p = 0$인 경우에 해당**



Information Set Decoding (ISD) Variants

- **다양한 버전의 ISD 들이 있지만, 큰 성능 차이가 존재하지 않음**



- **Syndrome decoding 문제를 해결하는데 효율적인 Quantum 버전의 Classical 알고리즘을 선택하는 것**
  - **Classical ≠ Quantum**
  - **Lee-Brickel**은 Weight check가 **2번**인 반면, **Prange**에서는 **1번**

Grover vs. McEliece

Daniel J. Bernstein *

The improvement introduced by Stern in [23] increases the number of roots by a larger factor. However, it also increases the cost of each iteration by more than the square root of the same factor. I do not see how Stern's collision-searching idea can save time in quantum information-set decoding.

Grover on ClassicMcEliece

Grover on ClassicMcEliece

- **제일 먼저 필요한 것은 중첩 상태의 Input**
  - 공개키 H에서 $(n-k)$ Columns을 선택하는 모든 경우의 수를 가지고 있는 중첩 상태의 $\psi(H_{(n-k)})$를 구성할 수 있어야 함

- $\psi(H_{(n-k)})$이 준비된다면, ISD 알고리즘의 수식을 양자 회로로 구현하여 적용 만 하면 됨

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

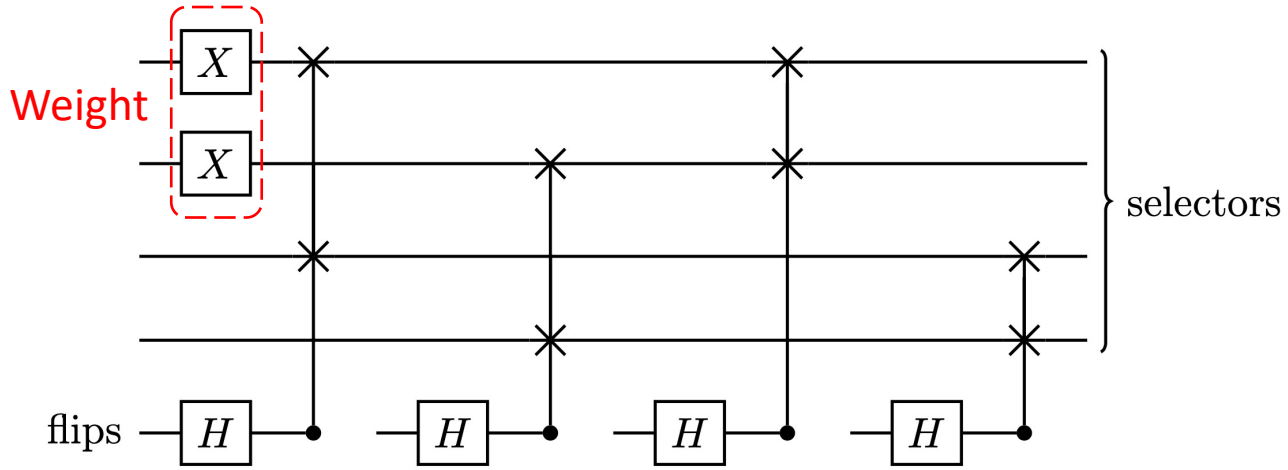
$$\boldsymbol{e} = (0000000011000000), (n = 16, t = 2)$$

Secret

$$\boldsymbol{C} = \boldsymbol{He} = (11010011)$$

Quantum Brute Force on CM (Butterfly Network)

- **Butterfly Network (BN)**을 활용한 n**-bit Vector 설정 (Input)**

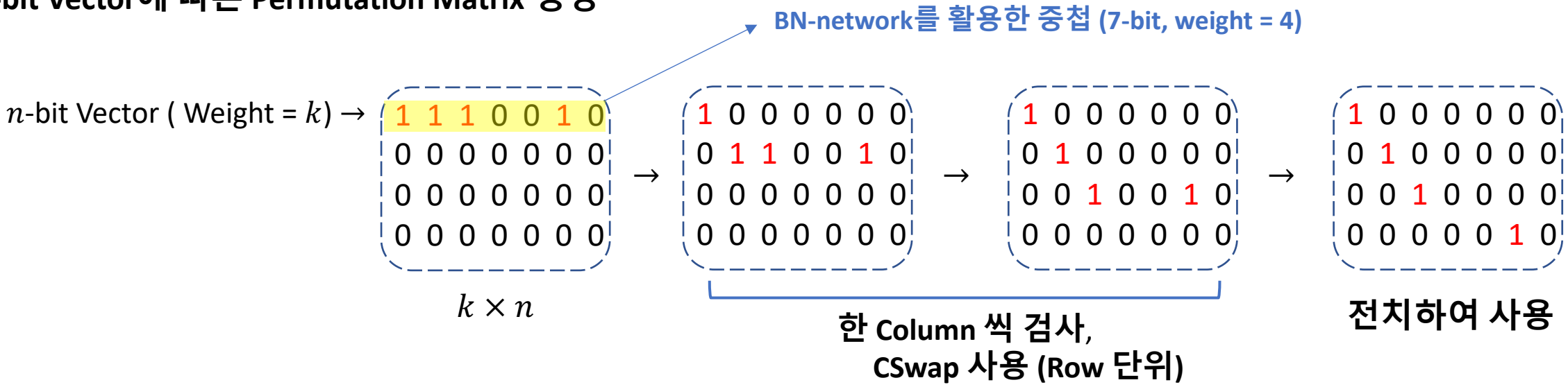


4-bit Vector (Weight = 2)

- **동일한 amplitude**를 위해 **CSwap당 하나의 큐비트**(flips) 할당

Grover on ClassicMcEliece

- **n-bit Vector에 따른 Permutation Matrix 생성**



$$H = \begin{matrix} 1\ 1\ 1\ 1\ 0\ 0\ 0 \\ 1\ 1\ 0\ 0\ 1\ 0\ 0 \\ 1\ 0\ 0\ 1\ 1\ 0\ 1 \\ 0\ 1\ 0\ 1\ 1\ 1\ 0 \end{matrix} \times \begin{matrix} 1\ 0\ 0\ 0\ 0\ 0\ 0 \\ 0\ 1\ 0\ 0\ 0\ 0\ 0 \\ 0\ 0\ 1\ 0\ 0\ 0\ 0 \\ 0\ 0\ 0\ 0\ 0\ 1\ 0 \end{matrix}^{T} = \begin{matrix} 1\ 1\ 1\ 0 \\ 1\ 1\ 0\ 0 \\ 1\ 0\ 0\ 0 \\ 0\ 1\ 0\ 1 \end{matrix}$$

$H_{(n-k)}$

```
def Grover_on_CM(eng):

    n = 16; k = 8 # k is (n-k)

    # Public key (Parity Check Matrix)
    h0 = eng.allocate_qureg(n); h1 = eng.allocate_qureg(n)
    h2 = eng.allocate_qureg(n); h3 = eng.allocate_qureg(n)
    h4 = eng.allocate_qureg(n); h5 = eng.allocate_qureg(n)
    h6 = eng.allocate_qureg(n); h7 = eng.allocate_qureg(n)

    # Ciphertext(Syndrome value)
    c = eng.allocate_qureg(k)

    # Permutation matrix(wieght k)
    perm = eng.allocate_qureg(k*n)

    # ISD target matrix
    hk = eng.allocate_qureg(k * k)

    # Temp qubits for ISD
    temp0 = eng.allocate_qureg(k)
    temp1 = eng.allocate_qureg(k)

    # Temp qubits for Weight Check
    weight_temp = eng.allocate_qureg(6)
    c0 = eng.allocate_qubit()
    c1 = eng.allocate_qubit()
```

H_k

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

$$e = (0000000011000000), (n = 16, t = 2)$$

Secret

$$C = He = (11010011)$$

$$\text{Perm} = \begin{pmatrix} 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \end{pmatrix}$$

BN target (Superposition)

```
# Set Ciphertext(Syndrome, Known value)
init(eng, c, 0b11010011, k)

init(eng, h0, 0b1000000010000111, n)
init(eng, h1, 0b0100000001110001, n)
init(eng, h2, 0b0010000011111011, n)
init(eng, h3, 0b0001000001011101, n)
init(eng, h4, 0b0000100000011110, n)
init(eng, h5, 0b0000010011001110, n)
init(eng, h6, 0b0000001010110100, n)
init(eng, h7, 0b0000000101100110, n)

# n-qubit vector(Weight-t) in superposition state,
# n = 16, t = 2, [n = 2^4, flips = 4 * (2^3) = 32]
#BN(eng, perm, flips, 4)
#X | perm[0]
#X | perm[1]

# Select k-Columns (Replace to BN-network)
X | perm[6]
X | perm[7]
X | perm[8]
X | perm[9]
X | perm[10]
X | perm[11]
X | perm[12]
X | perm[13]
```

큐비트 제한

$$H = \left(\begin{array}{cccccc|cc|cccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \end{array}\right)$$

$$C = He = (11010011)$$

```
# n-qubit vector(Weight-t) in superposition state,
# n = 16, t = 2, [n = 2^4, flips = 4 * (2^3) = 32]
#BN(eng, perm, flips, 4)
#X | perm[0]
#X | perm[1]

# Select k-Columns (Replace to BN-network)
X | perm[6]
X | perm[7]
X | perm[8]
X | perm[9]
X | perm[10]
X | perm[11]
X | perm[12]
X | perm[13]

print('Matrix H')
print_matrix(eng, h, n * k, n)

print('Selected k-columns (Superposition)')
print_state(eng, perm, 16)

Generate_Perm(eng, perm, k, n)
print('Permutation matrix')
print_matrix(eng, perm, k * n, n)

print('Matrix Hk')
Generate_hk(eng, h, perm, hk, k, n)
print_matrix(eng, hk, k*k, k)
```

```
/Users/kb/PycharmProjects/projectq/
Matrix H
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 1 0 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

Selected k-columns (Superposition)
0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0

Permutation matrix
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0

Matrix Hk
0 0 1 0 0 0 0 1

0 0 0 1 1 1 0 0

0 0 1 1 1 1 1 0

0 0 0 1 0 1 1 1

0 0 0 0 0 1 1 1

0 0 1 1 0 0 1 1

1 0 1 0 1 1 0 1

0 1 0 1 1 0 0 1
```

H

BN target (Superposition)

Permutation Matrix

$H_k = (H \times Perm^T)$

Grover on ClassicMcEliece

- **Quantum Gauss Jordan Elimination (2005), 이쪽에 관한 논문이 딱히 없는 것 같음, 2017년에 재 발표?**

QUANTUM GAUSS JORDAN ELIMINATION

DO NGOC DIEP[1] AND DO HOANG GIANG[2]

ABSTRACT. In this paper we construct the Quantum Gauß Jordan Elimination (QGJE) Algorithm and estimate the complexity time of computation of Reduced Row Echelon Form (RREF) of an $N \times N$ matrix using QGJE procedure. The main theorem asserts that QGJE has computation time of order $2^{N/2}$.

- **해당 논문에서는 Grover 알고리즘을 사용 → 반복 수행을 한다는 뜻,**
  - **Grover 안에서 한번 더 Grover? → 비효율적**

2. QGJE ALGORITHM

Step 1 Use the Grover's Search algorithm to find out the first non-zero $a_{i1} \neq 0$.
Step 2 If the search is successful, produce the first leading 1 in the first place as a_{11}, else change to the next column and repeat step 1.
Step 3 Eliminate all other entries $a_{1,1}, \ldots, a_{N,1}$ in the column.
Step 4 Change N to $N-1$, control if still $N > 0$, repeat the procedure from the step 1.
Step 5 In backward eliminate all $a_{N-1,N}, \ldots, a_{1,N}$.
Step 6 Check if $N > 0$, change N to $N-1$ and repeat the step 5.

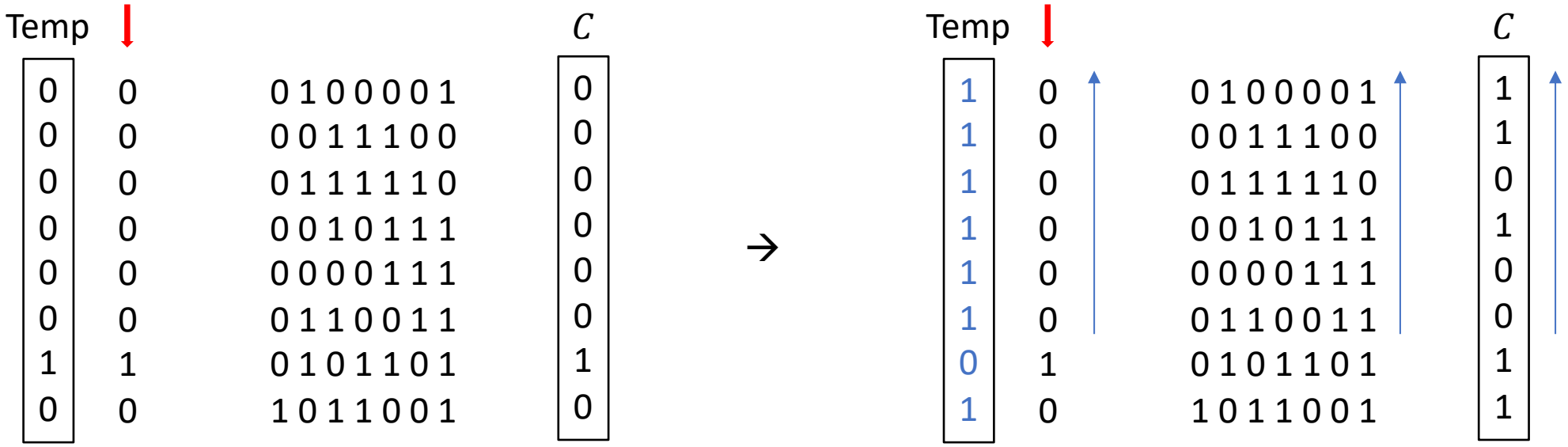
Grover on ClassicMcEliece

- **양자 게이트만으로 QGJE 구현**

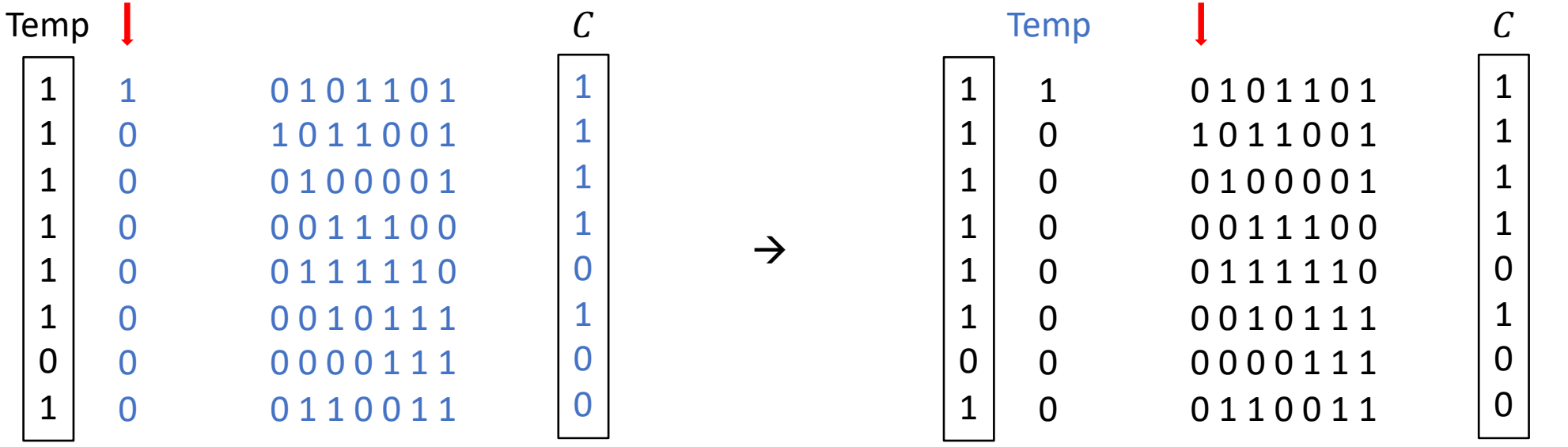
- **QISD에는 Inversion matrix** $(H_{n-k})^{-1}$를 생성 할 필요까지는 없음
  - $(H_{n-k})^{-1}$를 만드는데 **사용된 연산들**만 있으면 됨
  - 연산 과정 중, H_k 또한 일부분 훼손되어도 됨

- **Classical Gaussian Jordan** : **Swap 후 Elimination하여** 삼각 identity 행렬 구성

- **Quantum Gaussian Jordan** : Elimination 후 Rotation



- **All(X) 후, CSwap 반복으로 1이 최상위로 올 때 까지 Rotation, C에도 같이 수행**
- **Temp 역할 변경**



Temp		C
1	0 1 0 1 1 0 1	1
1	1 0 1 1 0 0 1	1
0	0 1 0 0 0 0 1	1
0	0 0 1 1 1 0 0	1
0	0 1 1 1 1 1 0	0
0	0 0 1 0 1 1 1	1
0	0 0 0 0 1 1 1	0
0	0 1 1 0 0 1 1	0

Temp		C
1	0 1 0 1 1 0 1	1
0	1 0 1 1 0 0 1	1
1	0 1 0 0 0 0 1	1
1	0 0 1 1 1 0 0	1
1	0 1 1 1 1 1 0	0
1	0 0 1 0 1 1 1	1
1	0 0 0 0 1 1 1	0
1	0 1 1 0 0 1 1	0

Temp		C
0	0 1 0 1 1 0 1	1
1	1 0 1 1 0 0 1	1
0	0 1 0 0 0 0 1	1
0	0 0 1 1 1 0 0	1
0	0 1 1 1 1 1 0	0
0	0 0 1 0 1 1 1	1
0	0 0 0 0 1 1 1	0
0	0 1 1 0 0 1 1	0

Temp		C
0	0 1 0 1 1 0 1	1
1	1 0 1 1 0 0 1	1
1	0 1 0 0 0 0 1	1
0	0 0 1 1 1 0 0	1
1	0 1 1 1 1 1 0	0
0	0 0 1 0 1 1 1	1
0	0 0 0 0 1 1 1	0
1	0 1 1 0 0 1 1	0

Temp		C
0	1 0 1 1 0 1	1
1	0 1 1 0 0 1	1
	↓	
0	1 0 0 0 0 1	1
1	0 1 1 1 0 0	1
0	0 1 1 1 1 1	1
1	0 1 0 1 1 1	1
1	0 0 0 1 1 1	0
0	0 1 0 0 1 0	1

Temp		C
0	1 0 1 1 0 1	1
1	0 1 1 0 0 1	1
0	1 0 0 0 0 1	
	↓	1
0	0 1 1 1 0 0	1
0	0 1 1 1 1 1	1
0	0 1 0 1 1 1	1
0	0 0 0 1 1 1	0
0	0 1 0 0 1 0	1

...

Temp		C
	←	
1	1 0 1 0 1 1 0 1 ↓	1
0	1 1 0 1 1 0 0 1	1
0	0 1 1 0 0 0 0 1	1
1	0 0 1 1 1 1 0 0	1
0	0 1 1 0 1 0 1 1	0
1	0 0 1 1 1 1 1 1	0
1	0 0 0 0 1 1 1 0	0
1	0 1 1 1 0 1 0 1	0

...

최종 C
0
0
1
1
0
0
0
0

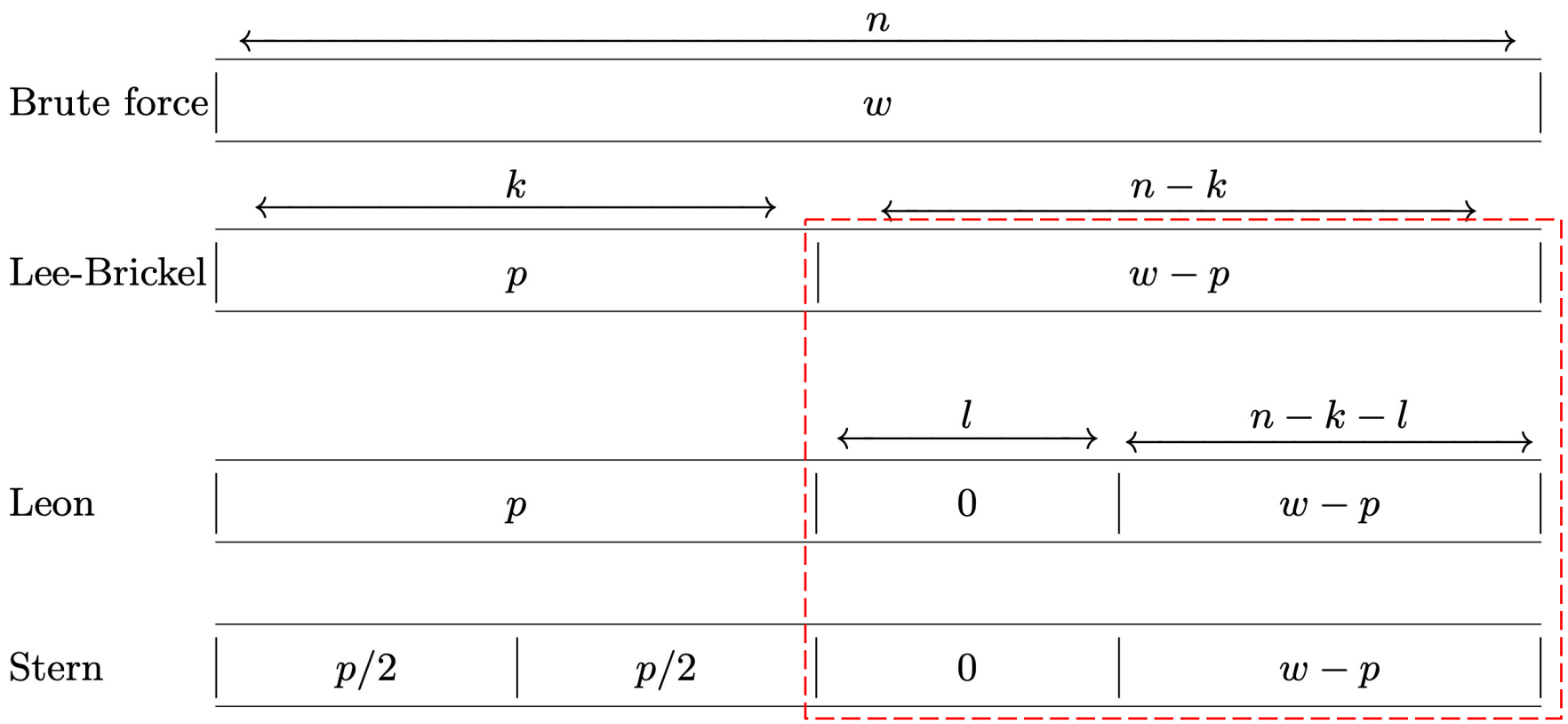
- 오른쪽에서 왼쪽은 Rotation 필요 없음
- Elimination만 수행
- Temp 큐비트 한번 더 할당
- 마지막으로 Weight check

$e = (0000000011000000)$ **Secret**

$C = He = (11010011)$

Information Set Decoding (ISD) Variants

- **Prange ISD는 k에서 $p = 0$인 경우에 해당**



실 행 결 과

Conclusion

- **Information Set Decoding(ISD)→코드기반암호에 대한 가장 효율적인 공격 알고리즘**

- Brute force 공격을 가속화 시키는 Grover search 알고리즘 적용 시, **공격 복잡도 √ 로 떨어짐**

- 다양한 Classical한 Information Set Decoding 알고리즘들의 성능이 Quantum으로 옮겼을 때
  → 동일한 성능을 보여주진 않음

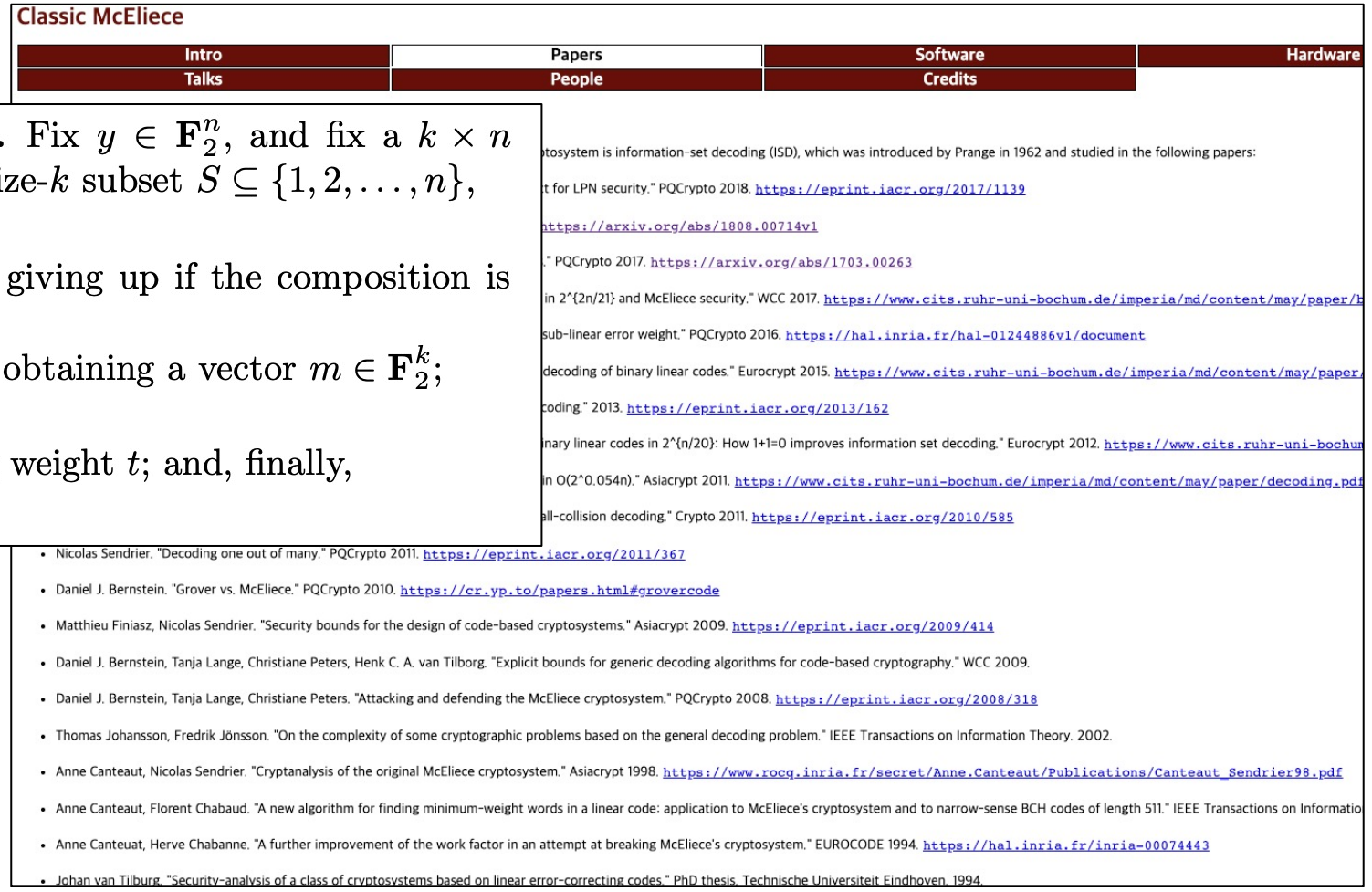
- **Quantum 버전의 최적화 된 ISD 알고리즘 구현**이 중요
  → **양자 자원 소모**를 따져야 함(Qubit, Quantum gates, Circuit Depth)

Conclusion

- **QISD 연구 동향**

Basic quantum information-set decoding. Fix $y \in \mathbf{F}_2^n$, and fix a $k \times n$ matrix G. Consider the function that, given a size-k subset $S \subseteq \{1, 2, \ldots, n\}$,

- inverts the composition $\mathbf{F}_2^k \xrightarrow{G} \mathbf{F}_2^n \to \mathbf{F}_2^S$, giving up if the composition is not invertible;
- applies the inverse to the image of y in $\mathbf{F}_2^S$, obtaining a vector $m \in \mathbf{F}_2^k$;
- computes $Gm \in \mathbf{F}_2^n$;
- gives up if $Gm - y$ does not have Hamming weight t; and, finally,
- returns 0.



Classic McEliece

Intro | Papers | Software | Hardware
Talks | People | Credits

tosystem is information-set decoding (ISD), which was introduced by Prange in 1962 and studied in the following papers:

t for LPN security." PQCrypto 2018. https://eprint.iacr.org/2017/1139

https://arxiv.org/abs/1808.00714v1

." PQCrypto 2017. https://arxiv.org/abs/1703.00263

in 2^{2n/21} and McEliece security." WCC 2017. https://www.cits.ruhr-uni-bochum.de/imperia/md/content/may/paper/b

sub-linear error weight." PQCrypto 2016. https://hal.inria.fr/hal-01244886v1/document

decoding of binary linear codes." Eurocrypt 2015. https://www.cits.ruhr-uni-bochum.de/imperia/md/content/may/paper/

coding." 2013. https://eprint.iacr.org/2013/162

nary linear codes in 2^{n/20}: How 1+1=0 improves information set decoding." Eurocrypt 2012. https://www.cits.ruhr-uni-bochun

in O(2^0.054n)." Asiacrypt 2011. https://www.cits.ruhr-uni-bochum.de/imperia/md/content/may/paper/decoding.pdf

all-collision decoding." Crypto 2011. https://eprint.iacr.org/2010/585

- Nicolas Sendrier. "Decoding one out of many." PQCrypto 2011. https://eprint.iacr.org/2011/367
- Daniel J. Bernstein. "Grover vs. McEliece." PQCrypto 2010. https://cr.yp.to/papers.html#grovercode
- Matthieu Finiasz, Nicolas Sendrier. "Security bounds for the design of code-based cryptosystems." Asiacrypt 2009. https://eprint.iacr.org/2009/414
- Daniel J. Bernstein, Tanja Lange, Christiane Peters, Henk C. A. van Tilborg. "Explicit bounds for generic decoding algorithms for code-based cryptography." WCC 2009.
- Daniel J. Bernstein, Tanja Lange, Christiane Peters. "Attacking and defending the McEliece cryptosystem." PQCrypto 2008. https://eprint.iacr.org/2008/318
- Thomas Johansson, Fredrik Jönsson. "On the complexity of some cryptographic problems based on the general decoding problem." IEEE Transactions on Information Theory. 2002.
- Anne Canteaut, Nicolas Sendrier. "Cryptanalysis of the original McEliece cryptosystem." Asiacrypt 1998. https://www.rocq.inria.fr/secret/Anne.Canteaut/Publications/Canteaut_Sendrier98.pdf
- Anne Canteaut, Florent Chabaud. "A new algorithm for finding minimum-weight words in a linear code: application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511." IEEE Transactions on Informatio
- Anne Canteuat, Herve Chabanne. "A further improvement of the work factor in an attempt at breaking McEliece's cryptosystem." EUROCODE 1994. https://hal.inria.fr/inria-00074443
- Johan van Tilburg. "Security-analysis of a class of cryptosystems based on linear error-correcting codes." PhD thesis. Technische Universiteit Eindhoven. 1994.

Conclusion

- 실제 Goppa 코드를 사용하는 Classic McEliece에 대한 Quantum Information Set Decoding를 구현 (8x16)

- 실제 CM 파라미터에 대한 자원 분석
  - mceliece348864 → (768 X 3488)
  - 코드는 파라미터만 바꾸면 동작해서 큰 문제는 X
  - BN-network 부분 최적화 및 Grover iteration 수 고려 필요