논문 20-45-12-02 The Journal of Korean Institute of Communications and Information Sciences '20-12 Vol.45 No.12
https://doi.org/10.7840/kics.2020.45.12.2037

LDPC 부호 기반 포스트 양자 암호 시스템을 위한 복호 알고리즘

김 정 현◆°

Decoding Algorithms for LDPC Code-Based Post Quantum Cryptosystems

Junghyun Kim◆°

요 약

양자 컴퓨터의 등장은 현재 가장 널리 사용되고 있는 공개키 암호 시스템에 심각한 위협이 되고 있다. 이러한 문제에 대처하기 위해 NIST는 2017년부터 포스트 양자 암호 표준화를 진행하고 있다. 통신 시스템의 대표적인 오류 정정 부호로 널리 사용되는 LDPC 부호는 NIST 표준화의 두 후보인 BIKE와 LEDAcrypt에서 부호 기반 암호 기법에 적용될 부호로 제안되었다. 부호 기반 암호 시스템에서 복호 성능은 암호 시스템의 안전성을 결정하기 때문에 매우 중요한 지표이다. 본 논문에서는 LDPC 부호 기반 포스트 양자 암호 시스템에서 사용되는 다양한 복호 기술을 소개하고 그 의미를 파악하여 다가오는 양자 컴퓨팅 시대를 대비하고자 한다.

Key Words : Post quantum cryptography, Low-density parity-check codes, Hard decision decoding, Bit flipping algorithm, Code-based cryptography

ABSTRACT

The emergence of practical quantum computers poses a significant threat to the most popular public key cryptosystems in current use. In order to cope with this problem, NIST has been standardizing post-quantum cryptography algorithms since 2017. The LDPC code, which is widely used as a representative error correction code for communication systems, has been proposed as a code to be applied to the code-based cryptography in BIKE and LEDAcrypt, two candidates for NIST standardization. In code-based cryptosystems, decoding performance is significant because it determines the security level of the cryptosystems. In this paper, we introduce the proposed decoding techniques in the LDPC code-based post quantum cryptosystems to prepare for the upcoming quantum computing era.

Ⅰ. 서 론

양자 컴퓨터 기술은 국제적 IT 기업들의 주도로 빠른 속도로 발전하고 있다. 최근에는 이미 특정 문제에 대해 슈퍼컴퓨터의 성능을 뛰어넘은 것으로 평가되고 있다. 따라서 현재 널리 사용되는 RSA 암호와 타원곡선 암호(ECC) 기반의 암호화 및 전자서명 알고리즘을 해독할 수 있는 수준의 양자 컴퓨터가 10년 이내에 등장할 것으로 예상된다[1].

양자 컴퓨터 기술이 발전함에 따라 기존 공개키 암

※ 이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2019R1G1A110000212).
◆° First and Corresponding Author : Soonchunhyang University, Department of Big Data Engineering, kimjh@sch.ac.kr, 조교수, 종신회원
논문번호 : 202008-206-A-RU, Received August 20, 2020; Revised September 18, 2020; Accepted September 24, 2020

호 시스템의 안전성에 문제가 제기되고 있다. 현재 전자상거래에 사용되는 공개키 암호 시스템들은 소인수 분해와 이산대수 등을 활용하여 개인키에 대한 정보 없이 해를 구하려면 수십 년 이상 걸리는 수학적 문제에 기반하여 동작하는데, 초고속 연산이 가능한 양자 컴퓨터가 개발된다면 이러한 암호 시스템은 커다란 위협을 받을 것으로 예상된다.

이러한 문제에 대응하기 위해서 2017년부터 미국의 NIST에서는 포스트 양자 암호 표준화를 단계적으로 진행하고 있다[2]. 2017년 11월에 제출이 마감된 1라운드에서는 총 67개의 알고리즘들이 평가 대상으로 선정되었고, 2019년 1월에 2라운드 진출 알고리즘으로 26개의 알고리즘들이 선정되었다[3]. 최근 2020년 7월에 3라운드 알고리즘으로 공개키 암호화 및 KEM 트랙에서 총 4개, 전자서명 트랙에서 총 3개의 알고리즘들이 선정되었다[4]. 또한 최종 선정된 7개 알고리즘들에 추가로 8개의 알고리즘들이 후보군으로 선정되어 향후 표준화에 선정될 수 있는 여지를 남겨두고 있다. 또 다른 특징으로는 각 부호는 모두 서로 다른 카테고리에 속하는 것을 볼 수 있으며, 양자 연산의 안전성을 보장하기 위해서 알고리즘의 다양성을 최대한 유지하려고 하는 의도가 보여진다. 3라운드 알고리즘들은 향후 저자들에 의해 한 번 더 수정할 수 있는 기회가 주어져 있기 때문에 2020년 이내에 수정된 문서 및 알고리즘이 다시 제안될 예정이다.

2라운드를 통과한 후보 알고리즘들 중 통신 시스템에서 널리 사용되는 대표적인 오류 정정 부호인 LDPC 부호[5,6]가 적용된 알고리즘으로 BIKE[7]와 LEDAcrypt[8]가 있다. 최근 3라운드에서는 이들 중 BIKE가 추가 후보군으로 선정되었다. LEDAcrypt는 최종 후보군에 포함되지는 못하였으나 LEDAcrypt에서 제안된 복호 알고리즘들은 BIKE 또는 다른 부호 기반 암호에 적용될 수 있으므로 여전히 중요한 의미를 갖는다. 따라서 본 논문에서는 BIKE와 LEDAcrypt에서 제안된 다양한 복호 알고리즘들에 대해 소개하고 그 의미를 살펴본다.

Ⅱ. LDPC 부호 기반 포스트 양자 암호의 복호

BIKE와 LEDAcrypt의 복호 방식으로 LDPC 부호의 경판정(hard decision) 복호를 위해 흔히 사용되는 비트 반전(bit flipping) 알고리즘[5,6]이 변형되어 제안되었다. 기본 Bit flipping 알고리즘은 수신 신드롬으로부터 암호문에 포함된 에러 벡터를 추정하고 추정한 에러 벡터가 적합한지 확인하는 과정으로 구성된다. 에러 벡터의 적합성을 판단하기 위하여 우선 수신한 신드롬으로부터 추정한 에러 벡터를 패리티 검사 행렬의 전치행렬에 곱하여 에러 벡터에 대한 신드롬을 구한다. 수신 신드롬에서 에러 벡터에 대한 신드롬을 이진 연산으로 더한 후 0이 아닌 값이 존재하지 않으면 추정한 에러 벡터가 적합하다고 판단할 수 있다. 기본 Bit flipping 알고리즘의 전체 동작은 알고리즘 1에 표현되어 있다.

알고리즘 1. Bit flipping 알고리즘

$e \leftarrow 0^n$
$s' \leftarrow s$
while $|s'| > 0$ **do**
  $\tau \leftarrow$ threshold $\in [0, 1]$
  for $j = 0, ..., n-1$ **do**
    if $|s' \cap h_j| \geq \tau |h_j|$ **then**
      $e[j] \leftarrow e[j] \oplus 1$
  $s' \leftarrow s \oplus eH^T$
return e

알고리즘이 시작되면 에러 벡터 e를 크기가 n이고 원소값이 모두 0인 벡터로 초기화하고 판정에 사용할 신드롬 s'을 수신한 신드롬 s으로 초기화한다. 이후 신드롬 s'에 0이 아닌 값이 존재하면 이하 과정을 반복한다. 우선 0에서 1사이의 경계값 τ를 설정한다. 모든 비트노드에 대하여 각 비트노드와 연결된 체크노드 중 체크식을 만족시키지 못하는 노드 수가 경계값 τ 비율보다 크거나 같으면 대응되는 에러 비트를 반전시킨다. 모든 비트노드에 대해 위 동작을 완료하면 수신 신드롬 s에 에러 벡터의 신드롬을 반영하여 새로운 신드롬 s'을 업데이트한다. 수정된 신드롬 s'의 모든 값이 0이 되면 최종 획득한 에러 벡터 e를 반환하고 알고리즘이 종료된다. 이하 본 논문에서 $\oplus$기호는 두 벡터의 각 원소 별 이진 덧셈 연산을 나타내고 $|x|$는 벡터 x에서 0이 아닌 원소의 개수를 나타낸다. h_j는 패리티 검사 행렬 H의 j번째 열벡터를 나타낸다.

2.1 BIKE에서 제안된 복호 알고리즘

BIKE에서는 One-round bit flipping 알고리즘과 Backflipping 알고리즘이 2라운드에 제안되었다. 3라운드에는 Black-Gray-Flip 알고리즘이 새롭게 제안되었다. One-round bit flipping 알고리즘은, 전체 비트노드가 비트 반전의 후보가 되는 기본 bit flipping 알고리즘과는 달리, 체크식을 만족시키지 못하는 체크노드가 상대적으로 많이 연결된 비트노드만 1차 비트 반전의 대상으로 고려된다. 또한 남은 에러를 처리하

알고리즘 2. One-round bit flipping 알고리즘

```
1: T ← threshold(|s|)
2: for j = 0, ..., n−1 do
3:    l ← min(ctr(H, s, j), T)
4:    J_l ← J_l ∪ j
5: e ← J_T
6: s' ← s ⊕ eH^T
7: while |s'| > S do
8:    for l = 0, ..., δ do
9:       e' ← check(H, s', J_{T−l}, d/2)
10:      (e, s') ← (e ⊕ e', s' ⊕ e'H^T)
11: e' ← check(H, s', e, d/2)
12: (e, s') ← (e ⊕ e', s' ⊕ e'H^T)
13: while |s'| > u do
14:   j ← guess_error_pos(H, s', d/2)
15:   (e[j], s') ← (e[j] ⊕ 1, s' ⊕ h_j)
16: return e

// threshold(S)
1: return function of r, w, t, S

// ctr(H, s, j)
1: return |s ∩ h_j|

// check(H, s, J, T)
1: e ← 0^n
2: for j ∈ J do
3:    if ctr(H, s, j) ≥ T then
4:       e[j] ← 1
5: return e

// guess_error_pos(H, s, T)
1: loop
2:    i ← Random sampling(s)
3:    for j ∈ eq_i do
4:       if ctr(H, s, j) ≥ T then
5:          return j
```

기 위하여 여전히 체크식을 만족시키지 못하는 체크노드에 연결된 비트노드에 대해 2차 비트 반전의 대상으로 고려하여 알고리즘을 수행한다. One-round bit flipping 알고리즘의 전체 동작은 알고리즘 2에 표현되어 있다.

알고리즘이 시작되면 수신한 신드롬 s에 대해 threshold 함수를 수행하여 경계값 T를 구한다. 모든 비트노드들에 대하여 ctr 함수를 수행하여 해당 비트노드와 연결된 체크노드들 중 체크식을 만족시키지 못하는 노드 수를 기준으로 비트노드 집합 J_l을 생성한다. 연결된 체크노드들 중 체크식을 만족시키지 못하는 노드 수가 경계값 T보다 크거나 같은 비트노드들(J_T에 속한 비트 노드들)은 에러 벡터에 포함시킨다. 수신한 신드롬 s에 대해 에러 벡터 신드롬을 더하여 수정된 신드롬 s'을 구한다.

수정된 신드롬 s'의 0이 아닌 값의 수가 기준값 S보다 큰 경우 다음을 반복한다. 각 비트노드 집합 J_l, $l = 0, ..., \delta$에 대하여 패리티 검사 행렬 H, 수정된 신드롬 s', 비트노드 집합 J_{T-l}, 경계값 $d/2$을 입력으로 check 함수를 수행하여 새로운 에러 벡터 e'을 구한다. 기존 에러 벡터 e에 새로운 에러 벡터 e'을 더하여 에러 벡터 e를 업데이트하고 수정된 신드롬 s'에 새로운 에러 벡터 e'의 신드롬을 더한 값으로 신드롬 s'을 업데이트한다. 수정된 신드롬 s'의 0이 아닌 값의 수가 기준값 S보다 작거나 같은 경우 위 반복이 종료된다. 이후 패리티 검사 행렬 H, 앞서 획득한 신드롬 s', 에러 벡터 e, 경계값 $d/2$를 입력으로 check 함수를 수행하여 새로운 에러 벡터 e'을 얻고 이를 기존 에러 벡터 e에 더하여 에러 벡터 e를 업데이트 한다. 또한 앞서 획득한 신드롬 s'에 새로운 에러 벡터 e'에 대한 신드롬을 더한 값으로 신드롬 s'을 업데이트 한다.

업데이트된 신드롬 s'의 0이 아닌 값의 수가 기준값 u보다 크면 다음 과정을 반복한다. 패리티 검사 행렬 H, 수정된 신드롬 s', 경계값 $d/2$을 입력으로 guess_error_pos 함수를 수행하여 비트노드의 인덱스를 추출한다. 추출된 비트노드에 대응되는 에러 비트 값과 연결된 체크노드의 값을 반전하여 신드롬 s'을 업데이트한다. 신드롬 s'의 0이 아닌 값의 수가 기준값 u보다 작거나 같으면 최종 획득한 에러 벡터 e를 반환하고 알고리즘이 종료된다.

각 내부 함수의 동작은 다음과 같다. threshold 함수는 신드롬을 입력으로 경계값을 반환하는 함수이다. ctr 함수는 패리티 검사 행렬 H, 신드롬 s, 비트 노드의 인덱스 j를 입력으로 받아 j번째 비트노드와 연결된 체크노드 중 체크식을 만족시키지 못하는 노드의 수를 반환하는 함수이다. check 함수는 패리티 검사 행렬 H, 신드롬 s, 비트노드의 인덱스 집합 J, 경계값 T를 입력으로 받아 비트노드 집합 J에 속하는 각 비트노드에 대하여 ctr 함수의 출력값이 경계값 T보다 큰 경우, 대응되는 에러 비트 값을 1로 설정하고 최종 에러 비트 벡터 e를 반환하는 함수이다. guess_error_pos 함수는 패리티 검사 행렬 H, 신드롬 s, 경계값 T를 입력으로 받아 신드롬에서 값이 1인 위치를 임의로 선택하고 선택된 위치에 대응되는 체크노드와 연결된 각 비트노드에 대하여 ctr 함수를 수행하여 결과값이 경계값 T보다 크거나 같으면 해당

알고리즘 3. Backflipping 알고리즘

$e \leftarrow 0^n$; $F \leftarrow 0^n$; $time \leftarrow 1$;
while $|s \oplus eH^T| > u$ **do**
  for j such that $F[j] = time$ **do**
    $e[j] \leftarrow e[j] \oplus 1$
    $F[j] \leftarrow 0$
  $s' \leftarrow s \oplus eH^T$
  $T \leftarrow$ threshold($|s'|, t - |F|$)
  $time \leftarrow time + 1$
  for $j \in 0, ..., n-1$ **do**
    if $|s' \cap h_j| \geq T$ **then**
      $e[j] \leftarrow e[j] \oplus 1$;
    $F[j] \leftarrow \begin{cases} 0, & F[j] \geq time \\ time + ttl(|s' \cap h_j| - T), & F[j] < time \end{cases}$
return e

비트노드의 인덱스를 반환하는 함수이다.

2라운드에서 One-round bit flipping 알고리즘과 함께 제안된 Backflipping 알고리즘의 전체 동작은 알고리즘 3에 표현되어 있다.

알고리즘이 시작되면 에러 벡터 e와 F 값을 크기가 n이고 원소값이 모두 0인 벡터로 초기화하고 현재 $time$을 1로 설정한다. 에러 벡터를 반영한 신드롬의 0이 아닌 값의 수가 기준값 u보다 크면 다음 과정을 반복한다. 각 비트노드의 F 값이 현재 $time$과 일치할 경우 해당 비트노드 위치의 에러 벡터 값을 반전시키고 F 값을 0으로 설정한다. 모든 비트노드에 대해 위 동작을 완료하면 수신 신드롬 s에 에러 벡터의 신드롬을 더하여 새로운 신드롬 s'을 구한다. 새롭게 구한 신드롬 s'과 F 벡터를 이용하여 새로운 경계값 T를 구하고 $time$을 1만큼 증가시킨다. 각 비트노드에 대하여 연결된 체크노드 중 체크식을 만족시키지 못하는 체크노드의 수가 경계값 T보다 크거나 같으면 해당 비트노드 위치의 에러 벡터 값을 반전시키고 F 값을 업데이트한다. 에러 벡터를 반영한 신드롬의 0이 아닌 값의 수가 기준값 u보다 작거나 같으면 최종 획득한 에러 벡터 e를 반환하고 알고리즘이 종료된다.

Backflipping 알고리즘은 특정 비트노드의 에러 비트 반전이 일어나면 비트값을 유지하는 수명을 의미하는 F 값이 설정되고 수명이 다될 때까지 반복 후에도 여전히 반복 종료 조건을 만족하지 못하면 수명이 다한 비트노드의 에러 비트값을 다시 원래대로 반전시킨다.

3라운드에서 새롭게 제안된 Black-Gray-Flip 알고리즘의 전체 동작은 알고리즘 4에 표현되어 있다.

알고리즘에 시작되면 에러 벡터 e를 크기가 n이고 원소값이 모두 0인 벡터로 초기화한다. 총 알고리즘

알고리즘 4. Black-Gray-Flip 알고리즘

$e \leftarrow 0^n$
for $i = 1, ..., N^{iter}$ **do**
  $s' \leftarrow s \oplus eH^T$
  $T \leftarrow$ threshold($|s'|, i$)
  $e, black, gray \leftarrow$ BFIter(s', e, T, H)
  if $i = 1$ **then**
    $s' \leftarrow s \oplus eH^T$
    $T \leftarrow (d+1)/2+1$
    $e \leftarrow$ BFMaskedIter($s', e, black, T, H$)
    $e \leftarrow$ BFMaskedIter($s', e, gray, T, H$)
$s' \leftarrow s \oplus eH^T$
if $|s'| = 0$ **then**
  return e
else
  return false

// procedure BFIter(s, e, T, H)
for $j = 0, ..., n-1$ **do**
  if $ctr(H, s, j) \geq T$ **then**
    $e[j] \leftarrow e[j] \oplus 1$
    $black[j] \leftarrow 1$
  else if $ctr(H, s, j) \geq T - \tau$ **then**
    $gray[j] \leftarrow 1$
return $e, black, gray$

// procedure BFMaskedIter($s, e, mask, T, H$)
for $j = 0, ..., n-1$ **do**
  if $ctr(H, s, j) \geq T$ **then**
    $e[j] \leftarrow e[j] \oplus mask[j]$
return e

// ctr(H, s, j)
return $|s \cap h_j|$

반복 횟수 N^{iter} 동안 다음 과정을 반복한다. 수신 신드롬 s에 에러 벡터의 신드롬을 더하여 새로운 신드롬 s'을 구한다. 새롭게 구한 신드롬 s'과 현재 반복 횟수를 이용하여 경계값 T를 정한다. BFIter 함수를 통해 에러 벡터 e, $black$, $gray$ 값을 구한다. 첫 번째 반복의 경우 앞서 구한 $black$ 값을 마스크값으로 이용하여 BFMaskedIter 함수를 통해 에러 벡터 e를 업데이트하고, 앞서 구한 $gray$ 값을 마스크값으로 이용하여 BFMaskedIter 함수를 통해 에러 벡터 e를 한 번 더 업데이트한다. 총 알고리즘 반복 횟수 동안 위 과정을 반복 후 수신 신드롬 s에 에러 벡터를 더하여 새로운 신드롬 s'을 구하고 새롭게 구한 신드롬 s'의 0이 아닌 값의 수가 0이면 최종 획득한 에러 벡터 e를 반환하고 0이 아닌 경우는 실패를 알리는 값을 반환한 뒤 알고리즘이 종료된다.

각 내부 함수의 동작은 다음과 같다. BFIter 함수는

수신한 신드롬 s, 에러 벡터 e, 경계값 T, 패리티 검사 행렬 H를 입력으로 받아 모든 비트노드에 대하여 각 비트노드와 연결된 체크노드 중 체크식을 만족시키지 못하는 노드의 수가 경계값 T보다 크거나 같으면 대응되는 에러 비트의 값을 반전시키고 대응되는 $black$ 값을 1로 설정한다. 만일 비트노드와 연결된 체크노드 중 체크식을 만족시키지 못하는 노드의 수가 경계값 T보다 작고 $T-\tau$보다 크거나 같으면 $gray$ 값을 1로 설정한다. 모든 비트노드에 대하여 위 과정을 수행한 뒤 에러 벡터 e, $black$, $gray$ 값을 반환한다. BFMaskedIter 함수는 수신 신드롬 s, 에러 벡터 e, 마스크값 $mask$, 경계값 T, 패리티 검사 행렬 H를 입력으로 받아 모든 비트노드에 대하여 각 비트노드와 연결된 체크노드 중 체크식을 만족시키지 못하는 노드의 수가 경계값 T보다 크거나 같으면 대응되는 에러 비트의 값에 해당 마스크값을 이진 연산으로 더한다. 모든 비트노드에 대하여 위 과정을 수행한 뒤 에러 벡터 e를 반환한다.

2.2 LEDAcrypt에서 제안된 복호 알고리즘

LEDAcrypt에서는 Q-decoding 알고리즘이 2라운드에 제안되었고 LEDA decoder 알고리즘이 3라운드에 새롭게 제안되었다. LEDAcrypt의 패리티 검사 행렬은 두 행렬 H와 Q의 곱으로 생성된다. 이때 사용되는 두 행렬 H와 Q가 BIKE에서 사용되는 패리티 검사 행렬에 비해 상대적으로 희소(sparse)하기 때문에 실제 알고리즘 동작에서는 효율성을 고려하여 두 행렬의 전치행렬에서 원소값이 1인 위치를 차수(degree)로 표현한 Htr과 Qtr을 사용한다. 또한 패리티 검사 행렬이 두 행렬의 곱으로 표현되기 때문에 비트노드와 체크노드 사이에 중간노드가 새롭게 정의된다. Q-decoding 알고리즘의 전체 동작은 알고리즘 5에 표현되어 있다.

알고리즘 5. Q-decoding 알고리즘

```
1: iter ← 0
2: repeat
3:   unsat_pc ← 0^{n_0 p}
4:   s' ← s
5:   for i = 0 to n_0 − 1 do
6:     for exp = 0 to p − 1 do
7:       for h = 0 to d_v − 1 do
8:         if s'[(exp + Htr[i][h]) mod p] = 1 then
9:           unsat_pc[i·p + exp] ← unsat_pc[i·p + exp] + 1
10:  w̄ ← MAX({w | (w, th) ∈ LutS and w < |s'|})
11:  th̄ ← th | (w̄, th) ∈ LutS
12:  for i = 0 to n_0 − 1 do
13:    for exp = 0 to p − 1 do
14:      similarity ← 0
15:      k_st ← 0
16:      for i' = 0 to n_0 − 1 do
17:        for k = k_st to Σ_{j=0}^{i'} m_j − 1 do
18:          qrow[k] ← i'·p + ((exp + Qtr[i][k]) mod p)
19:          similarity ← similarity + unsat_pc[qrow[k]]
20:        k_st ← Σ_{j=0}^{i'} m_j
21:      if similarity ≥ th̄ then
22:        e[i·p + exp] ← e[i·p + exp] ⊕ 1
23:        for k = 0 to m − 1 do
24:          for h = 0 to d_v − 1 do
25:            a = ⌊qrow[k]/p⌋ ;   b = qrow[k] mod p;
26:            idx ← (Htr[a][h] + b) mod p
27:            s[idx] ← s[idx] ⊕ 1
28:  iter ← iter + 1
29: until |s| > 0 and iter < imax
30: if |s| = 0 then
31:   return e, true
32: else
33:   return e, false
```

알고리즘 5에서 현재 반복 횟수 iter를 0으로 설정하고 다음 과정을 반복한다. 모든 중간노드에 대하여 각 중간노드에 연결된 체크노드 중 대응되는 신드롬의 값이 1인(즉, 체크식을 만족시키지 못한) 체크노드의 수를 해당 중간노드의 unsat_pc 값으로 설정한다. 룩업 테이블 LutS에서 새롭게 구한 신드롬의 0이 아닌 값의 수보다 작은 w값들 중 가장 큰 값을 $\overline{w}$로 설정하고 $\overline{w}$에 대응되는 th값을 $\overline{th}$로 설정한다. 모든 비트노드에 대해 각 비트노드에 연결된 모든 중간노드의 unsat_pc 값을 더한 값을 해당 비트노드의 similarity 값으로 설정한다. 만일 similarity 값이 $\overline{th}$보다 크거나 같으면 해당 비트노드에 대응되는 에러 비트 값을 반전시키고, 반전된 에러 비트 값을 신드롬에 반영시킨다. 모든 비트노드에 대하여 위 과정을 수행한 후 현재 반복 횟수 iter를 1만큼 증가시킨다.

신드롬의 0이 아닌 값의 수가 0보다 크거나 현재 반복 횟수 iter가 최대 반복 횟수 imax보다 작은 횟수 동안 위 과정을 반복한다. 반복이 끝난 후 신드롬에서 0이 아닌 값의 수가 0이면 에러 벡터 e와 true 값을 반환하고, 0보다 크면 에러 벡터 e와 false 값을 반환한 뒤 알고리즘이 종료된다.

3라운드에서 제안된 LEDA decoder 알고리즘에서는 특별히 행렬 Q가 단위행렬인 경우를 가정한다. 따라서 Htr만 사용된다. LEDA decoder 알고리즘의 전체 동작은 알고리즘 6에 표현되어 있다. 알고리즘이 시작되면 에러 벡터 e를 크기가 n_0p이고 원소값이 모

두 0인 벡터로 초기화한다. 최대 내부 반복 횟수 imax^{In} 동안 신드롬 s의 값들 중 0이 아닌 값이 존재하지 않으면 반복을 멈추고 존재하면 RandomizedInPlaceIter 함수를 통해 에러 벡터 e와 신드롬 s를 업데이트한다. 내부 반복이 종료되면 외부 반복이 시작된다. 최대 외부 반복 횟수 imax^{Out} 동안 신드롬 s의 값들 중 0이 아닌 값이 존재하지 않으면 반복을 멈추고 존재하면 OutOfPlaceIter 함수를 통해 에러 벡터 e와 신드롬 s를 업데이트한다. 외부 반복이 종료되면 신드롬 s의 0이 아닌 값의 수를 판별하여 0인 경우는 에러 벡터 e와 true 값을 반환하고, 0보다 큰 경우는 에러 벡터 e와 false 값을 반환한 뒤 알고리즘이 종료된다.

내부 함수인 RandomizedInPlaceIter 함수는 에러 벡터 e, 패리티 검사 행렬의 전치 행렬 Htr, 신드롬 s, 현재 내부 반복 횟수 iter^{In}을 입력으로 에러 벡터 e와 신드롬 s를 업데이트하는 함수이다. 먼저 출력 에러 벡터 $\bar{e}$와 출력 신드롬 $\bar{s}$를 입력 에러 벡터 e와 입력 신드롬 s로 초기화한다. 현재 내부 반복 횟수 iter^{In}과 입력 신드롬 s를 입력으로 경계값 th를 계산한다. 임의의 순서대로 비트노드 인덱스 집합 J를 생성하고 J의 원소 순서대로 대응되는 비트노드에 대하여 해당 비트노드와 연결된 체크노드 중 체크식을 만족시키지 못하는 노드의 수를 upc값으로 설정한다. 설정된 upc값이 경계값 th보다 크거나 같으면 대응되는 출력 에러 비트 값을 반전시키고 대응되는 출력 신드롬 값도 반전시킨다. 모든 비트노드에 대하여 위 과정이 완료되면 출력 에러 벡터 $\bar{e}$와 출력 신드롬 $\bar{s}$를 반환한다.

또 다른 내부 함수인 OutOfPlaceIter 함수는 에러 벡터 e, 패리티 검사 행렬의 전치 행렬 Htr, 신드롬 s, 현재 외부 반복 횟수 iter^{Out}을 입력으로 에러 벡터 e와 신드롬 s를 업데이트하는 함수이다. 먼저 출력 에러 벡터 $\bar{e}$와 출력 신드롬 $\bar{s}$를 입력 에러 벡터 e와 입력 신드롬 s로 초기화한다. 현재 외부 반복 횟수 iter^{Out}과 입력 신드롬 s를 입력으로 경계값 th를 계산한다. 모든 비트노드에 대하여 각 비트노드와 연결된 체크노드 중 체크식을 만족시키지 못하는 노드의 수를 upc값으로 설정한다. 설정된 upc값이 경계값 th보다 크거나 같으면 대응되는 출력 에러 비트 값을 반전시키고 대응되는 출력 신드롬 값도 반전시킨다. 모든 비트노드에 대하여 위 과정이 완료되면 출력 에러 벡터 $\bar{e}$와 출력 신드롬 $\bar{s}$를 반환한다.

알고리즘 6. LEDA decoder 알고리즘

```
e ← 0^{n_0 p}
for iter^In ← 0 to imax^In − 1 do
  if |s| = 0 then break
  {e, s} ← RandomizedInPlaceIter(e, Htr, s, iter^In)
for iter^Out ← 0 to imax^Out − 1 do
  if |s| = 0 then break
  {e, s} ← OutOfPlaceIter(e, Htr, s, iter^Out)
if |s| = 0 then
  return e, true
else
  return e, false

// RandomizedInPlaceIter
ē ← e; s̄ ← s; th ← ComputeThreshold(iter^In, s);
π ← Random permutation of size n_0 p
J ← π(⟨0, 1, ..., n_0 p − 1⟩)
for j ∈ J do
  i ← ⌊ j/p ⌋ ; offset ← j mod p; upc[j] ← 0;
  for h = 0 to d_v − 1 do
    if s[(Htr[i][h] + offset) mod p] = 1 then
      upc[j] ← upc[j] + 1
  if upc[j] ≥ th then
    ē[j] ← ē[j] ⊕ 1
    for h = 0 to d_v − 1 do
      idx ← (Htr[i][h] + offset) mod p
      s̄[idx] ← s̄[idx] ⊕ 1
return {ē, s̄}

// OutOfPlaceIter
ē ← e; s̄ ← s; th ← ComputeThreshold(iter^Out, s);
for i = 0 to n_0 − 1 do
  for offset = 0 to p − 1 do
    j ← i · p + offset
    upc[j] ← 0
    for h = 0 to d_v − 1 do
      if s[(Htr[i][h] + offset) mod p] = 1 then
        upc[j] ← upc[j] + 1
    if upc[j] ≥ th then
      ē[j] ← ē[j] ⊕ 1
      for h = 0 to d_v − 1 do
        idx ← (Htr[i][h] + offset) mod p
        s̄[idx] ← s̄[idx] ⊕ 1
return {ē, s̄}
```

Ⅲ. 성능비교 및 분석

기본 Bit flipping 알고리즘과 BIKE에서 제안하는 One-round bit flipping 알고리즘, Backflipping 알고리즘, Black-Gray-Flip 알고리즘과 LEDAcrypt에서 제안하는 Q-decoding 알고리즘, 그리고 LEDA decoder 알고리즘에 대해 성능비교 및 분석을 수행하

였다. 패리티 검사 행렬로 행의 무게가 10이고 256행 512열 크기의 이진 행렬을 랜덤하게 생성하여 사용하였다. 이하 논문에서 각 알고리즘을 BF, ORBF, BFBF, BGFBF, LEDA-Q, LEDA-L로 표시한다. 실험환경으로 에러 벡터의 무게를 변화시키면서 알고리즘들의 복호 실패 비율을 측정하였다.

BF에서 성능을 좌우하는 파라미터인 τ는 각 비트노드에 연결된 체크노드 중 체크식을 만족시키지 못하는 체크노드의 비율에 따라 비트 반전 여부를 결정하는 기준값이다. 따라서 0에서 1사이의 값으로 설정되는데 그 크기에 따라 성능 차이가 매우 큼을 확인하였다. 그림 1은 $\tau=1$과 $\tau=0.5$인 경우의 성능을 비교한 결과이다.

ORBF에서 성능을 좌우하는 파라미터인 δ는 실험을 통해 5로 설정하였다. 랜덤 샘플링 기법으로는 신드롬 값이 1인 위치에 대응되는 체크노드를 랜덤하게 선택한 후 해당 체크노드에 연결된 비트노드를 랜덤하게 선택하는 방법(method-1), 각 비트노드에 연결된 체크식을 만족시키지 못하는 체크노드의 수가 경계값보다 큰 비트노드를 랜덤하게 선택하는 방법(method-2), 각 비트노드에 연결된 체크식을 만족시키지 못하는 체크노드의 수가 가장 큰 비트노드 중 랜덤하게 선택하는 방법(method-3)을 고려하였다. 실험을 통해 세 번째 방식이 가장 좋은 성능을 가짐을 확인하였다. 이는 만족되지 못한 여러 체크식에 연결된 비트노드는 에러가 발생했을 확률이 높으므로 해당 비트를 우선적으로 반전시켜주는 것이 전체 복호에 좋은 영향을 미치기 때문이다. 그림 2는 세 가지 랜덤 샘플링 기법에 따른 성능을 비교한 결과이다.

BFBF에서 성능을 좌우하는 ttl 함수[7]는 다음과 같이 정의된다.

$$ttl(\delta)=\max(1,\min(5,\lfloor \alpha\delta+\beta \rfloor)) \quad (1)$$

여기서 α와 β는 반전된 비트가 상태를 유지할 반복 횟수를 입력값 δ에 따라 조절해주는 상수값들이다. 또한 1과 5는 반전된 비트를 최소 1번에서 최대 5번의 반복 동안 유지하기 위한 값이다. 이러한 값들을 잘못 설정하면 반전되었던 비트가 모든 체크식이 만족되기 전에 다시 원래 상태로 재반전되기 때문에 시스템 환경에 따른 적절한 설정이 매우 중요하다. α와 β는 패리티 검사 행렬의 행의 무게, 에러 벡터의 무게, 복호기의 최대 반복 횟수에 영향을 받는다. 위 세 가지 값들이 상대적으로 큰 경우는 α값을 β값에 비해 작게 설정하는 것이 성능개선에 효과적임을 확인하였다. 그림 4는 $\alpha=0.45$, $\beta=1.1$인 경우와 $\alpha=1$, $\beta=1$인 경우에 대해 성능을 비교한 결과이다. 본 논

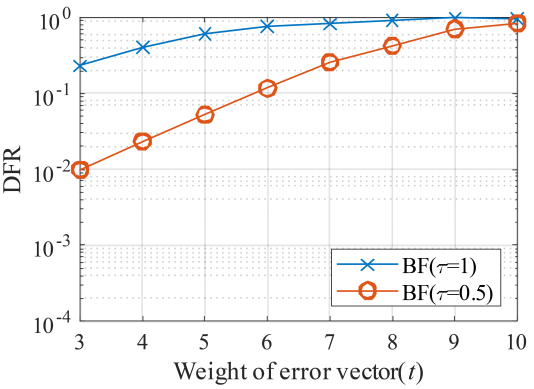


그림 1. BF의 에러 벡터 무게에 따른 복호 실패율.
Fig. 1. Decoding failure ratio versus weight of error vector for BF.

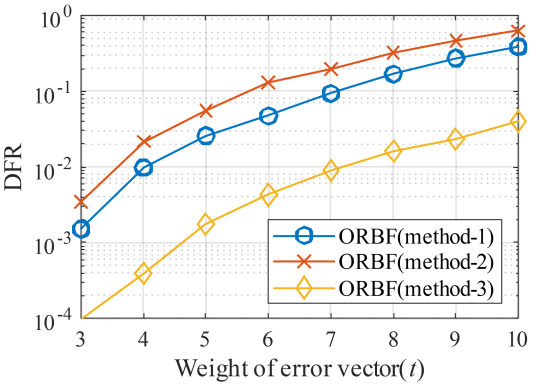


그림 2. ORBF의 에러 벡터 무게에 따른 복호 실패율.
Fig. 2. Decoding failure ratio versus weight of error vector for OFBF.

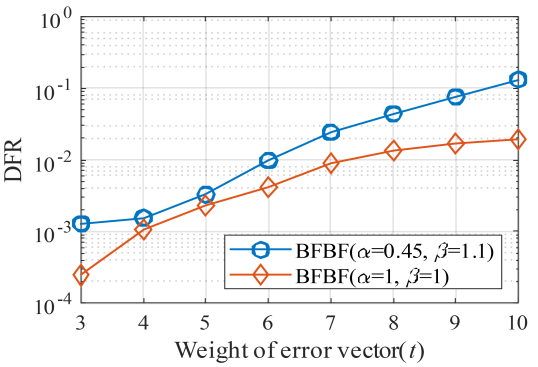


그림 3. BFBF의 에러 벡터 무게에 따른 복호 실패율.
Fig. 3. Decoding failure ratio versus weight of error vector for BFBF.

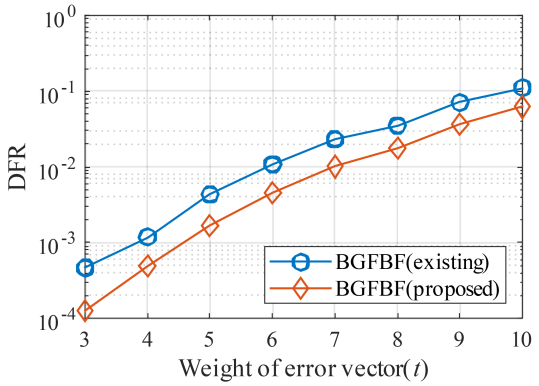


그림 4. BGFBF의 에러 벡터 무게에 따른 복호 실패율.
Fig. 4. Decoding failure ratio versus weight of error vector for BGFBF.

문에서 설정한 실험환경은 패리티 검사 행렬의 무게와 에러 벡터의 무게가 상대적으로 작은 환경에 해당하므로 $\alpha=1$, $\beta=1$인 경우가 더 좋은 성능을 보임을 확인할 수 있다. 또한 $\alpha=0.45$, $\beta=1.1$인 경우 오류 벡터 무게가 작은 영역에서 오류 마루 현상이 나타났다. 이는 α값이 상대적으로 작으면 반전된 비트가 전체 체크식이 만족될 때까지 유지되지 못하고 다시 이전 상태로 복구되는 경우가 발생할 수 있기 때문이다.

본 논문에서는 BGFBF의 성능개선을 위하여 기존 알고리즘 4에서 8행을 삭제하고 4행과 5행 사이에 다음을 추가한 형태의 알고리즘을 제안한다.

$$T \leftarrow \max\left(\left\lfloor \frac{d}{2} \right\rfloor + 1, T\right) \tag{2}$$

이는 초기 반복 시 기존 알고리즘보다 크거나 같은 경계값을 적용하여 반전 후보를 설정함으로써 오류 확률이 높은 비트를 우선적으로 반전시키는 효과를 준다. 또한 반복 복호 과정에서 과도하게 경계값이 작아져서 오류 확률이 낮은 비트들이 반전되는 것을 막아준다. 이러한 개선을 통해 알고리즘 성능을 향상시킬 수 있다. 그림 4는 기존 BGFBF와 제안한 BGFBF에 대해 성능을 비교한 결과이다. 에러 벡터 무게에 대하여 전 범위에서 제안하는 기법이 기존 기법 대비 좋은 성능을 보임을 확인할 수 있다.

기존 LEDA-Q 알고리즘은 경계값 설정에 따라 성능이 좌우된다. 만약 경계값이 너무 크게 설정되면 모든 체크식을 만족시키기 전에 비트 반전 없이 반복 복호만 진행되는 경우가 발생할 수 있다. 이를 보완하기 위하여 비트 반전 조건을 만족시키는 비트노드가 없을 경우 각 비트노드에 연결된 체크식을 만족시키지 못하는 체크노드의 수가 가장 큰 비트노드 중 랜덤하게 선택하는 방법을 알고리즘 5의 27행 이하에 추가하여 제안한다. 그림 5는 기존 LEDA-Q와 제안한 LEDA-Q에 대해 성능을 비교한 결과이다. 에러 벡터 무게에 대하여 전 범위에서 제안하는 기법이 기존 기법 대비 탁월한 성능을 보임을 확인할 수 있다.

본 논문의 실험환경에서 가장 좋은 성능을 보이는 파라미터 설정을 적용한 각 알고리즘들의 성능을 그림 6에 표현하였다. BF, ORBF, BFBF, LEDA-Q는 최대 반복 횟수를 100회로 설정하고, BGFBF, LEDA-L은 5회로 설정하였다. 실험결과 BF와 제안한 LEDA-Q를 제외한 모든 기법들의 성능 차이는 근소하게 나타났다. 최대 반복 횟수 측면에서는 제안한

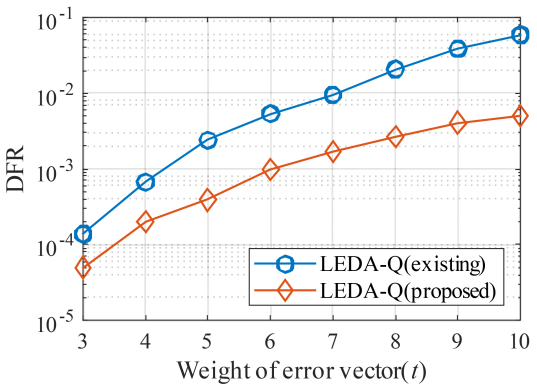


그림 5. LEDA-Q의 에러 벡터 무게에 따른 복호 실패율.
Fig. 5. Decoding failure ratio versus weight of error vector for LEDA-Q.

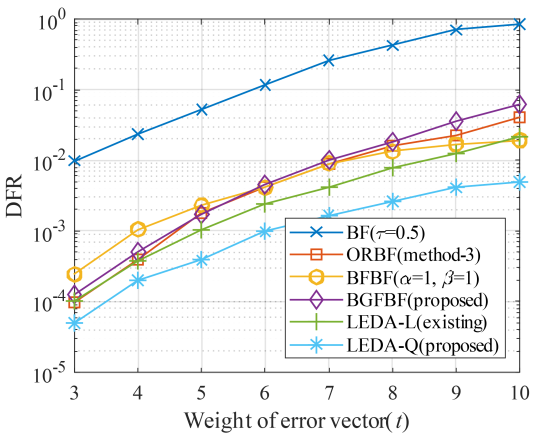


그림 6. BIKE와 LEDAcrypt 복호 알고리즘들의 에러 벡터 무게에 따른 복호 실패율.
Fig. 6. Decoding failure ratio versus weight of error vector for decoding algorithms of BIKE and LEDAcrypt.

BGFBF와 LEDA-L이 우수하고, 성능 측면에서는 제안한 LEDA-Q가 탁월함을 확인할 수 있다.

Ⅳ. 결 론

본 논문에서는 현재 NIST에서 진행 중인 포스트 양자 암호 표준화를 위해 제안된 LDPC 부호 기반의 포스트 양자 암호 시스템에서 사용되는 다양한 복호 알고리즘들을 소개하였다. 소개된 알고리즘들은 성능 향상을 위하여 기본 비트 반전 알고리즘을 다양한 방식으로 개선하였다. 그러나 실험결과에서 살펴보았듯이 각 알고리즘 간의 성능 차이가 크지 않기 때문에 향후 표준화 과정에서 고려되는 다양한 파라미터와 시스템 환경에서 실험을 통한 성능 평가 및 비교 연구는 앞으로 활발히 이루어질 것으로 예측된다. 향후 연구 주제로 각 알고리즘의 최적화 기법, 복잡도 분석, 순차적 복호 방식의 적용 등을 고려할 수 있다.

References

[1] I. Mustafa, I. U. Khan, S. Aslam, A. Sajid, S. M. Mohsin, M. Awais, and M. B. Qureshi, “A lightweight post-quantum lattice-based RSA for secure communications,” *IEEE Access*, vol. 8, pp. 99273-99285, 2020.

[2] NIST, *Post-Quantum Cryptography*(PQC), https://csrc.nist.gov/projects/post-quantum-cryptography

[3] NIST, *NIST PQC Round 2 Submisstions*, https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions

[4] NIST, *NIST PQC Round 3 Submisstions*, https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions

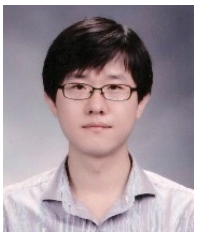
[5] R. G. Gallager, “*Low density parity check codes*,” MIT Press, 1963.

[6] N. Miladinovic and M. Fossorier, “Improved bit-flipping decoding of low-density parity-check codes,” *IEEE Trans. Inf. Theory*, vol. 51, no. 4, pp. 1594-1606, Apr. 2005.

[7] BIKE, *BIKE - Bit Flipping Key Encapsulation*, https://bikesuite.org

[8] LEDAcrypt, *LEDAcrypt: Low-dEnsity parity-check coDe-bAsed cryptographic systems*, https://www.ledacrypt.org

김 정 현 (Junghyun Kim)



2006년 8월 : 연세대학교 전기전자공학과 졸업

2008년 8월 : 연세대학교 전기전자공학과 석사

2010년 7월~2013년 2월 : 한국전자통신연구원 연구원

2017년 8월 : 연세대학교 전기전자공학과 박사

2017년 9월~2019년 2월 : 삼성전자 삼성리서치 책임연구원

2019년 3월~현재 : 순천향대학교 빅데이터공학과 조교수

<관심분야> 인공지능, 빅데이터, 무선통신시스템

[ORCID:0000-0003-0265-5169]