Implementation of SMAUG-T on AArch64 – Lattice

https://youtu.be/Na0iMklvWd8

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB

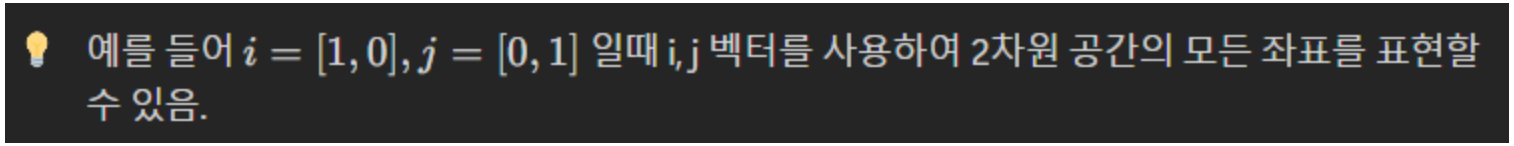
1. 격자 기반

- 선형 조합
  - 선형 조합은 숫자와 벡터를 곱하고 더하는 것.
  - $c_1 b_1 + c_2 b_2$와 같이 두 벡터(b)와 정수 또는 실수를 곱하고 벡터끼리 더해서 새로운 조합을 만드는 것.

- 선형 독립
  - 한 벡터를 다른 벡터들의 선형 조합으로 표현할 수 없음.
  - 즉, $b_1 = [1,0]$이고 $b_2 = [0,1]$ 이면 b_1 으로 b_2를 만들 수 없음. 따라서 두 벡터는 선형 독립이라고 함.
  - 반대로 $b_1 = [1,1]$ $b_2 = [2,2]$ 라고 하면 b_1으로 b_2 를 만들 수 있기 때문에 선형 종속이라고 함.

1. 격자 기반

- 기저 벡터(basis Vector)
  - 벡터 공간이나 격자를 생성하는 데 사용하는 선형 독립 벡터
  - 기저벡터는 두가지 중요한 성질이 필요
    - 선형 독립성 – 기저 벡터들은 서로 선형 독립이여야 함. 하나의 벡터로 다른 벡터들의 선형 조합을 만들 수 없어야 함.
    - 생성 능력 – 기적 벡터들은 그 공간의 모든 벡터를 선형 조합으로 표현할 수 있어야 함



1. 격자 기반

- 격자에서의 기저 벡터
  - $L = \{ c_1 b_1 + c_2 b_2 + \ ... + c_n b_n \mid c_i \ \in \mathbb{Z} \}$
  - 여기에서 $b_1, b_2 \ ..., b_n$은 기저 벡터
    - 기저 벡터라는 것은 곧 선형 독립인 벡터를 의미
  - 기저벡터 b_i와 c_i인 정수 계수 조합으로 격자 L의 모든 점을 생성할 수 있음

  - 격자 기반 암호에서 기저 벡터의 따라서 암호의 안전성이 높아지고, 더 작은 파라미터로 더 안전한 알고리즘을 설계할 수 있음

$$L = \{c_1 b_1 + c_2 b_2 \mid c_1, c_2 \in \mathbb{Z}\}$$

여기서 L은 b_1과 b_2로 생성된 격자를 나타내며, 이 격자의 모든 점은 c_1과 c_2라는 정수 계수를 사용해 표현될 수 있습니다.

예를 들어, $b_1 = [0, 1]$, $b_2 = [1, 0]$일 때:

- $c_1 = 2$, $c_2 = 3$이라면, $c_1 b_1 + c_2 b_2 = 2[0, 1] + 3[1, 0] = [3, 2]$가 됩니다.
- 따라서 격자는 정수 계수의 조합으로 격자 공간의 모든 점을 생성합니다.

1. 격자 기반

	n	k	q	η_1	η_2	(d_u, d_v)	δ
KYBER512	256	2	3329	3	2	(10, 4)	2^{-139}
KYBER768	256	3	3329	2	2	(10, 4)	2^{-164}
KYBER1024	256	4	3329	2	2	(11, 5)	2^{-174}

- 격자 기반 암호(LWE)의 매개변수
  - n : 기저 격자의 차원을 의미하는 매개변수.
    - 연산 성능과 보안성과 관련된 매개변수

- n is set to 256 because the goal is to encapsulate keys with 256 bits of entropy (i.e., use a plaintext size of 256 bits in KYBER.CPAPKE.Enc). Smaller values of n would require to encode multiple key bits into one polynomial coefficient, which requires lower noise levels and therefore lowers security. Larger values of n would reduce the capability to easily scale security via parameter k.
- k is selected to fix the lattice dimension as a multiple of n; changing k is the main mechanism in KYBER to scale security (and as a consequence, efficiency) to different levels.

  - q : 모듈러 값으로 사용하는 소수
    - 계산의 크기와 관련되어 있는 매개변수
    - 값이 커지면 더 넓은 공간에서 계산이 이루어져 안정성이 높아지지만 메모리 사용량이 증가하고 계산량이 많아짐

- We choose q as a small prime satisfying $n \mid (q-1)$; this is required to enable the fast NTT-based multiplication. There are two smaller primes for which this property holds, namely 257 and 769. However, for those primes we would not be able to achieve negligible failure probability required for CCA security, so we chose the next largest, i.e., $q = 3329$.

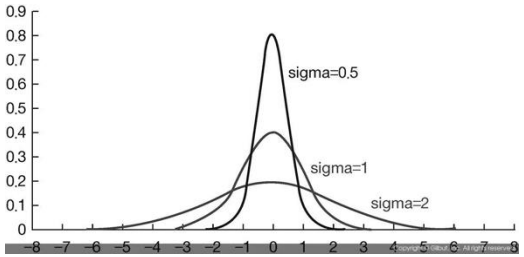
  - σ : 노이즈 크기와 관련된 매개변수
    - 값이 크면 노이즈가 커져서 공격자로부터 더 안전하지만 복호화 오류 가능성이 높아지는 문제가 있음

- The remaining parameters η_1, η_2, d_u and d_v were chosen to balance between security (see Section 4), ciphertext size, and failure probability. Note that all three parameter sets achieve a failure probability of $< 2^{-128}$ with some margin. We discuss this in more detail in Subsections 1.5 and 5.5.
- The parameter η_1 defines the noise of $\mathbf{s}$ and $\mathbf{e}$ in Algorithm 4 and of $\mathbf{r}$ in Algorithm 5. The parameter η_2 defines the noise of $\mathbf{e}_1$ and e_2 in Algorithm 5. We discuss the reason for setting $\eta_1 > \eta_2$ in the KYBER512 parameter set in Section 1.5.

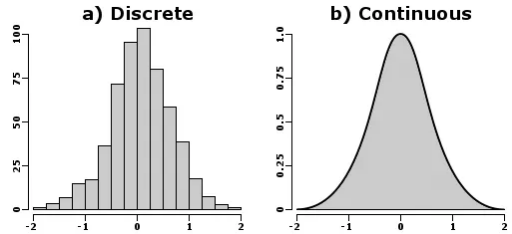
1. 격자 기반

- 노이즈 e 벡터의 역할은 암호문이 원래 메시지와 직접적인 연관성이 없도록 만드는 역할
  - LWE나 MLWE같은 문제에서 $b = As + e \, mod \, q$일때 e는 노이즈 벡터를 의미

- 노이즈 분포
  - 노이즈 벡터는 특정한 분포를 따르도록 설정됨. 가장 많이 사용되는 분포는 가우시안 분포(Gaussizan distribution)와 디스크리트 가우시안 분포(Discrete Gaussian Distribution)이다.
  - 가우시안 분포 – 정규 분포라고도 하며, 대부분이 평균값 근처에 집중됨. 표준편차가 분포의 폭을 결정하고 표준 편차가 크면 노이즈 값이 더 넓게 퍼지고 작으면 평균 값 근처에 집중됨.



1. 격자 기반



- 노이즈분포
  - 디스크리트 가우시안 분포 – 이산 가우시안 분포로 정수 값들만 가질 수 있는 형태. 연속적인 가우시안 분포를 정수 값들로 제한한 형태로, 컴퓨터에서는 정수 값 연산이 더 간단하고 효율적이기 때문에 컴퓨터에서 사용하기 좋음.

- 근데, 가우시안 분포는 비효율적이거나 부채널 공격에 취약하다고 함.
  - Kyber의 경우에는 Centered binomial 분포를 사용한다고 함.

Binomial noise. Theoretic treatments of LWE-based encryption typically consider LWE with Gaussian noise, either rounded Gaussian [94] or discrete Gaussian [28]. As a result, many early implementations also sampled noise from a discrete Gaussian distribution, which turns out to be either fairly inefficient (see, for example, [25]) or vulnerable to timing attacks (see, for example, [29, 89, 45]). The performance of the best known attacks against LWE-based encryption does not depend on the exact distribution of noise, but rather on the standard deviation (and potentially the entropy). This motivates the use of noise distributions that we can easily, efficiently, and securely sample from. One example is the centered binomial distribution used in [10]. Another example is the use of "learning-with-rounding" (LWR), which adds deterministic uniform noise by dropping bits as in KYBER's $\mathsf{Compress}_q$ function. In the design of KYBER we decided to use centered binomial noise and thus rely on LWE instead of LWR as the underlying problem.

1. 격자 기반

- $\mathbb{Z}_q[x]/f(x)$ 모듈러 다항식 링
  - 다항식 링은 다항식으로 구성된 집합.
  - q는 모듈러 값으로 $\mathbb{Z}_q[x]$ 에 포함되는 다항식의 계수가 0~(q-1) 사이의 값만 가질 수 있음.
  - $\mathbb{Z}_q[x]$에 포함되는 다항식은 $f(x)$로 나눠진 나머지 값임. 즉, $f(x)$보다 큰 다항식을 가질 수 없음.

예시로 이해하기:

다항식 링 $\mathbb{Z}_q[x]$는 **정수 계수**를 가진 다항식들로 이루어진 집합인데, 여기서 q는 **모듈러 값**이에요. 예를 들어:

- $f(x) = 3x^2 + 2x + 1$
- $g(x) = x^3 + 5x + 4$

이 다항식들은 모두 $\mathbb{Z}_7[x]$에 속할 수 있어요. 여기서 $\mathbb{Z}_7$는 모듈러 연산이 적용돼, 모든 계수 값이 0에서 6 사이의 값만 가질 수 있어요.

모듈러 다항식 링:

격자 기반 암호에서는 **모듈러 다항식 링** $\mathbb{Z}_q[x]/(f(x))$를 많이 사용해요. 여기서 $f(x)$는 **모듈러 다항식**이고, 이 다항식으로 나눈 나머지로 연산이 이루어져요.

예를 들어, $f(x) = x^4 + 1$이라면, $h(x) = x^5 + 3x + 2$는 다음과 같이 나눌 수 있어요:

$$h(x) \mod (x^4 + 1) = x + 3x + 2 = 4x + 2$$

1. 격자 기반

- RLWE(Ring-LWE)
  - LWE 문제를 다항식 링 구조로 확장한 형태
  - 데이터를 다항식으로 표현하고 다항식 연산을 통해 효율적으로 암호화 연산을 수행함.

수학적 정의

$a(x),\ s(x),\ e(x)$: 다항식 링 $\mathbb{Z}_q[x]/(f(x))$에 속하는 다항식

$b(x)$: 암호문 다항식

$$b(x) = a(x) \cdot s(x) + e(x)\ mod\ (q, f(x))$$

- $\mathbb{Z}_q[x]/(f(x))$: 모듈러 다항식 링. q는 모듈러 값, $f(x)$는 모듈러 다항식.
- $a(x)$: 공개된 랜덤 다항식
- $s(x)$: 비밀
- $e(x)$: 노이즈 다항식

  - 다항식 곱셈을 FFT, NTT 등의 알고리즘으로 최적화하여 효율적인 연산 가능
  - 작은 키 크기로도 높은 보안을 제공

1. 격자 기반

- MLWE(Module-LWE)
  - 모듈 구조를 사용해 여러 개의 작은 다항식 링 요소들을 결합해 모듈 행렬을 만들어 사용
    - RLWE는 다항식 하나를 사용했다면, MLWE는 다항식 여러 개를 갖고 있는 행렬을 사용

수학적 정의

A: $m \times n$ 크기의 모듈 행렬 (각 요소는 다항식 링 $\mathbb{Z}_q[x]/(f(x))$에 속함)

s: n 차원의 비밀 벡터(다항식 요소)

e: m 차원 노이즈 벡터 (다항식 요소)

$$b = A \cdot s + e \; mod \; (q, f(x))$$

A는 모듈 구조를 가진 행렬

s, e는 비밀 벡터와 노이즈 벡터

1. 격자 기반

	n	k	q	η_1	η_2	(d_u, d_v)	δ
KYBER512	256	2	3329	3	2	$(10, 4)$	2^{-139}
KYBER768	256	3	3329	2	2	$(10, 4)$	2^{-164}
KYBER1024	256	4	3329	2	2	$(11, 5)$	2^{-174}

알고리즘	기반문제	다항식 링	Degree n	Modulus q
Crystals-Kyber	Module LWE	$\mathbf{Z_q[X]/< X^n + 1 >}$	256	3329 (12-bit)
Crystals-Dilithium	Module LWE	$\mathbf{Z_q[X]/< X^n + 1 >}$	256	8380417 (23-bit)
Falcon	NTRU Lattice	$\mathbf{Z_q[X]/< X^n + 1 >}$	512, 1024	12289 (14-bit)

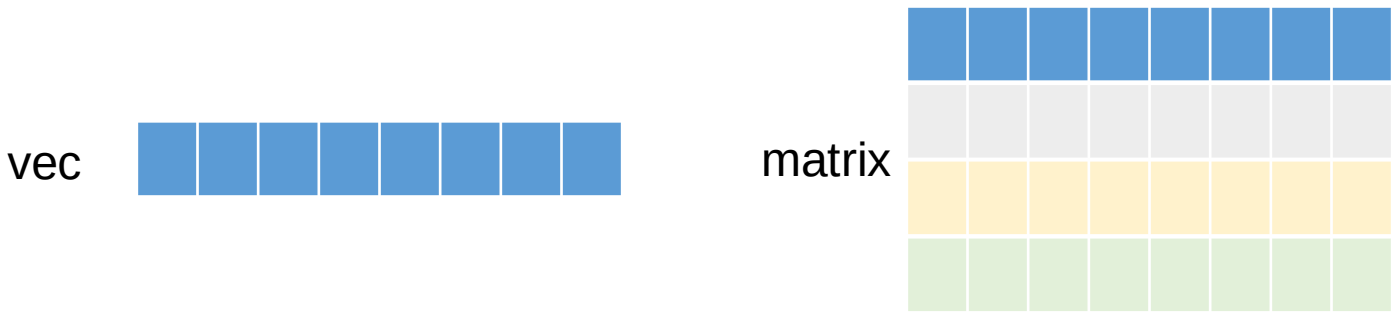
알고리즘	기반문제	다항식 링	Degree n	Modulus q	NTT Friendliness
SMAUG-T	Module LWE/LWR	$\mathbf{Z_q[X]/< X^n + 1 >}$	256	256, 512	No (Karatsuba, TC)
NTRU+	NTRU	$\mathbf{Z_q[X]/< X^n - X^{\frac{n}{2}} + 1 >}$	576, 768, 864, 1152	3457 (12-bit)	YES (Radix 2/3)
HAETAE	Module LWE	$\mathbf{Z_q[X]/< X^n + 1 >}$	256	64513 (16-bit)	YES (Radix 2)
NCC-Sign	Ring-LWE	$\mathbf{Z_q[X]/< X^n - X^{\frac{n}{2}} + 1 >}$	1152, 1536, 2304 (2048)	8401537, 8397313, 8404993 (8380417) (24-bit)	YES (Radix 2/3)
		$\mathbf{Z_q[X]/< X^n - X + 1 >}$	1201, 1607, 2039	17279291, 17305741, 17287423 (25-bit)	No (Karatsuba, TC)

1. 격자 기반

```
typedef struct {
    int16_t coeffs[LWE_N];
} poly;

typedef struct {
    poly vec[MODULE_RANK];
} polyvec;
```

- 매트릭스와 벡터의 차이
  - poly는 다항식을 의미함. 따라서 struct poly는 다항식의 계수를 가지고 있는 구조체.
    - 즉, $f(x) = a^0x^0 + a^1x^1 + \cdots + a^nx^n$ 같은 다항식이 있을 때, a^0, a^1, a^2 … 과 같은 계수의 값을 가지고 있는 배열
      - poly.coeffs = [a^0, a^1, a^2 …]
  - 여기서 벡터(vec)은 coeffs라고 볼 수 있음. 즉 1차원 데이터로 구성됨
  - 매트릭스는 2차원 배열임 즉, 행과 열로 구성되어 있음.
    - Polyvec은 f(x)와 같은 다항식을 여러 개 가지고 있는 2차원 배열임.
      - polyvec.vec = [[ploy.coeffs] , [poly.coeffs] …]



1. 격자 기반

```
inline void poly_add(poly *r, const poly *a,
    unsigned int i;
    for (i = 0; i < LWE_N; i++)
        r->coeffs[i] = a->coeffs[i] + b->coe
}
```

```
inline void poly_sub(poly *r, const poly *a,
    unsigned int i;
    for (i = 0; i < LWE_N; i++)
        r->coeffs[i] = a->coeffs[i] - b->coe
}
```

```
void vec_vec_mult(poly *r, const polyvec *a, const polyvec *b) {
    unsigned int i;
    for (i = 0; i < MODULE_RANK; i++)
        poly_mul_acc(a->vec[i].coeffs, b->vec[i].coeffs, r->coeffs);
}
```

```
void vec_vec_mult_add(poly *r, const polyvec *a, const polyvec *b,
                      const uint8_t mod) {
    unsigned int i, j;
    polyvec al;
    poly res;

    for (i = 0; i < MODULE_RANK; ++i)
        for (j = 0; j < LWE_N; ++j)
            al.vec[i].coeffs[j] = a->vec[i].coeffs[j] >> mod;

    memset(&res, 0, sizeof(poly));
    vec_vec_mult(&res, &al, b);
    for (j = 0; j < LWE_N; ++j)
        res.coeffs[j] <<= mod;

    poly_add(r, r, &res);
}
```

1. 격자 기반

```
void matrix_vec_mult_add(polyvec *r, const polyvec a[MODULE_RANK],
                         const polyvec *b) {
    unsigned int i, j, k;
    polyvec at;

    for (i = 0; i < MODULE_RANK; ++
        for (j = 0; j < MODULE_RANK
            for (k = 0; k < LWE_N;
                at.vec[j].coeffs[k]

        vec_vec_mult(&r->vec[i], &a
        for (j = 0; j < LWE_N; ++j)
            r->vec[i].coeffs[j] <<=
    }
}
```

```
void matrix_vec_mult_sub(polyvec *r, const polyvec a[MODULE_RANK],
                         const polyvec *b) {
    unsigned int i, j, k;
    polyvec al;
    poly res;

    for (i = 0; i < MODULE_RANK; ++i) {
        for (j = 0; j < MODULE_RANK; ++j)
            for (k = 0; k < LWE_N; ++k)
                al.vec[j].coeffs[k] = a[i].vec[j].coeffs[k] >> _16_LOG_Q;

        memset(&res, 0, sizeof(poly));
        vec_vec_mult(&res, &al, b);
        for (j = 0; j < LWE_N; ++j)
            res.coeffs[j] <<= _16_LOG_Q;

        poly_sub(&r->vec[i], &r->vec[i], &res);
    }
}
```

감 사 합 니 다