KYBER의 NTT 제거 가능성 분석

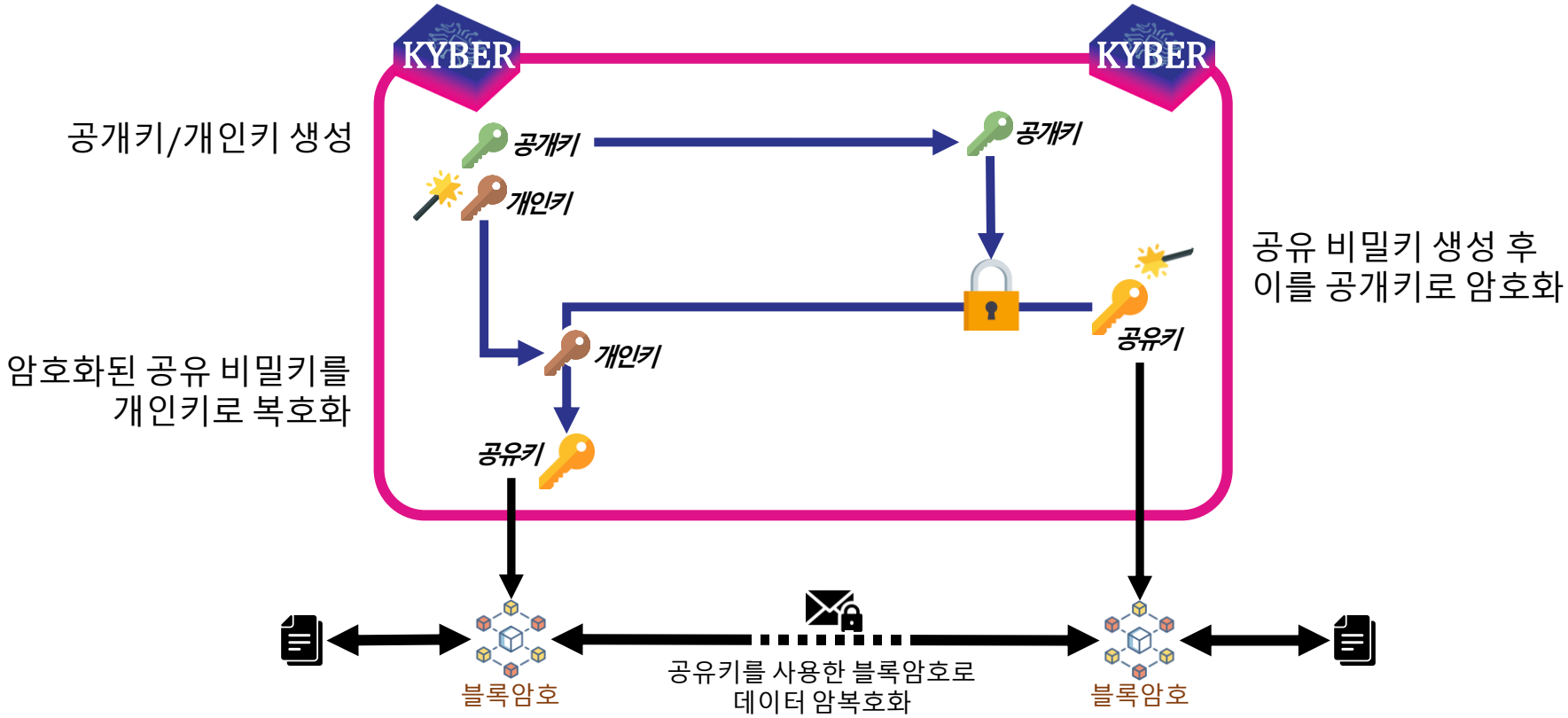
240111

국민대학교 류지은

INDEX

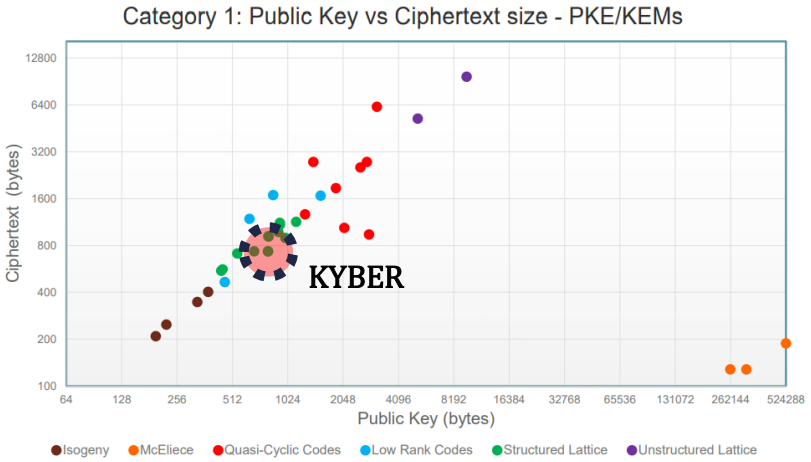
2005년 Oded Regev에 의하여 제안되어 발전한 KEM(Key Encapsulation Mechanism) 알고리즘

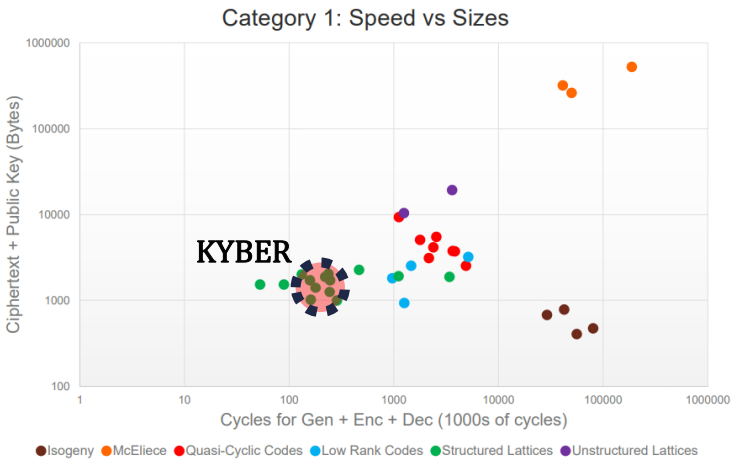
KEM 두 party가 데이터 암호화에 사용되는 비밀키를 공유하기 위하여 공개키 알고리즘을 활용하여 키를 생성 및 교환하는 과정



2017년부터 진행된 NIST PQC 공모전에서 2022년 7월 KEM 표준으로 선정됨

파라미터 및 암호문의 크기와 동작 속도의 측면에서 격자 기반 암호시스템으로서 이점을 가짐





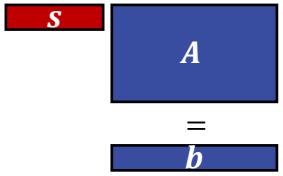
NIST PQC 공모전 3라운드 후보 KEM 알고리즘 비교

	KYBER	NTRU	SABER
기반문제	LWE(learning with error)	NTRU	LWR(learning with rounding)
다항식 환	$R = Z_q[x]/(x^n + 1)$	$R = Z[x]/(x^n - 1)$	$R = Z_q[x]/(x^n + 1)$
다항식 곱셈 알고리즘	NTT	합성곱	Toom-Cook, Karatsuba

그림 출처) D. Moody, "The 2nd round of the NIST PQC standardization process", In the second PQC standardization Conference, 2019.

LWE(learning with Error)

Learning without error



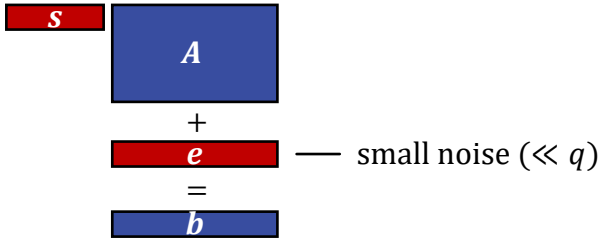
$$sk = \boldsymbol{s} = \{x, y, z, w\} = \{5, 50, 82, 10\}$$

$$\begin{cases} c_{1,1}x + c_{1,2}y + c_{1,3}z + c_{1,4}w = c_1 \\ \vdots \\ c_{n,1}x + c_{n,2}y + c_{n,3}z + c_{n,4}w = c_n \end{cases}$$

$$pk = \begin{pmatrix} \boldsymbol{A} = \{c_{1,1}, \dots, c_{n,4}\}, \\ \boldsymbol{b} = \{c_1, \dots, c_n\} \end{pmatrix}$$

pk가 주어지면 가우스 소거법을 활용하여 쉽게 sk를 구할 수 있음

Learning with error



$$\begin{cases} c_{1,1}x + c_{1,2}y + c_{1,3}z + c_{1,4}w + e_1 = c_1 + e_1 \\ \vdots \\ c_{n,1}x + c_{n,2}y + c_{n,3}z + c_{n,4}w + e_n = c_n + e_n \end{cases}$$

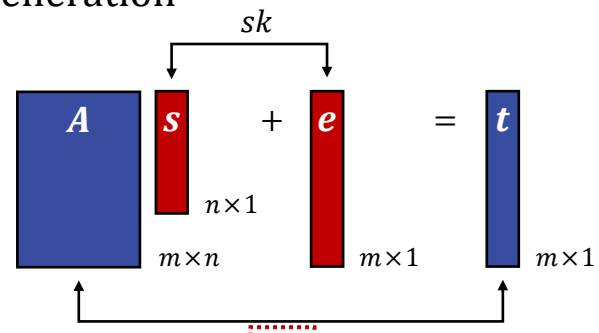
$$pk = \begin{pmatrix} \boldsymbol{A} = \{c_{1,1}, \dots, c_{n,4}\}, \\ \boldsymbol{b} = \{c_1 + e_1, \dots, c_n + e_n\} \end{pmatrix}$$

pk가 주어져도 sk를 구하기 어려움

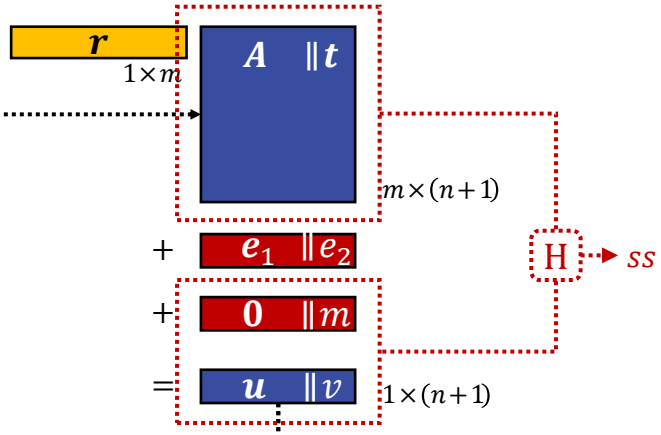
Alice

Bob

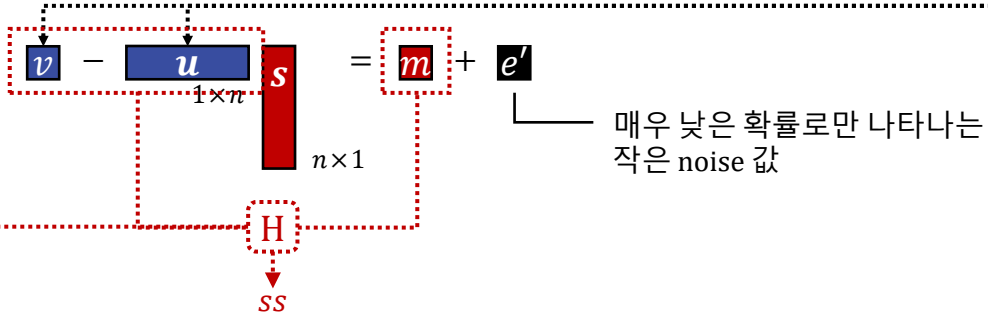
Key generation



Encryption

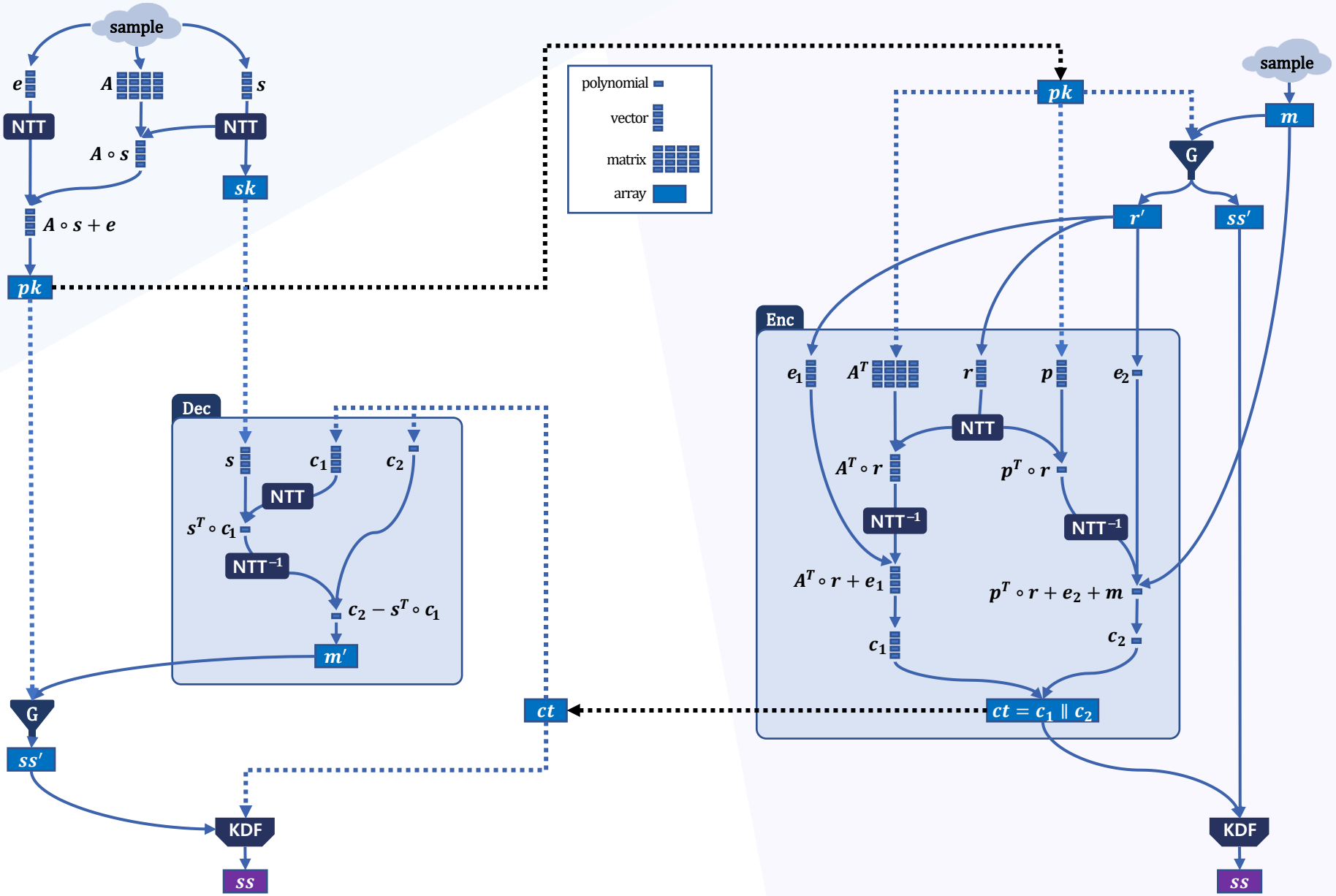


Decryption

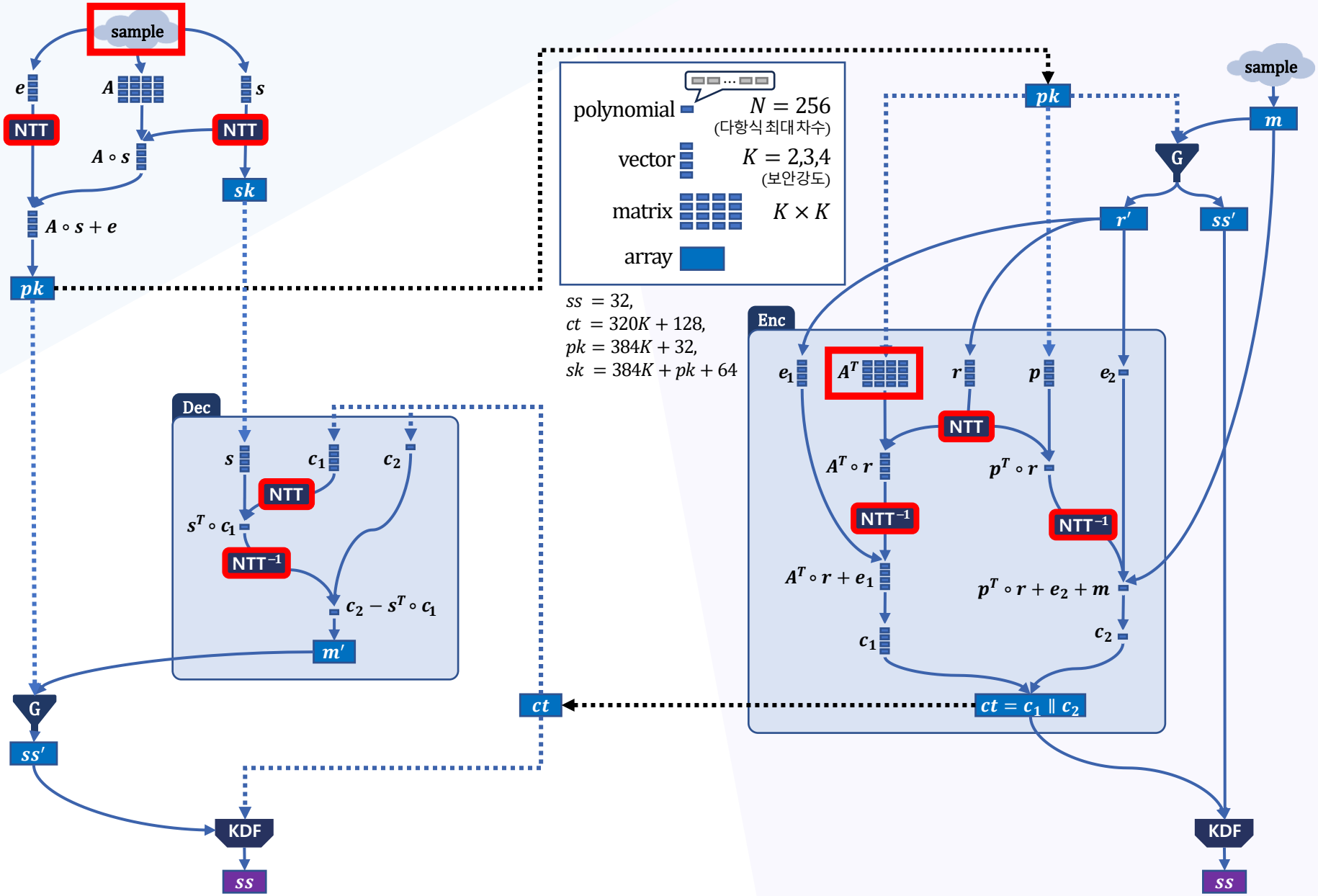


매우 낮은 확률로만 나타나는
작은 noise 값

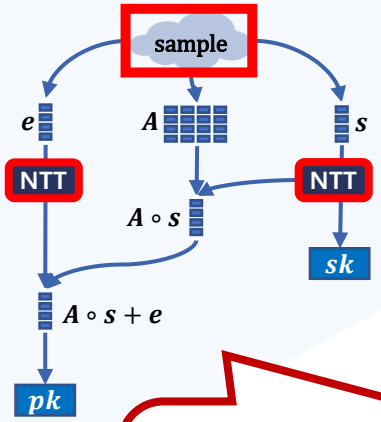
KYBER 개요


sample
e
A
s
NTT
NTT
A ∘ s
sk
A ∘ s + e
pk
polynomial
vector
matrix
array
pk
sample
m
G
r′
ss′
Enc
e1
A^T
r
p
e2
NTT
A^T ∘ r
p^T ∘ r
NTT^-1
NTT^-1
A^T ∘ r + e1
p^T ∘ r + e2 + m
c1
c2
ct = c1 ‖ c2
ct
Dec
s
c1
c2
NTT
s^T ∘ c1
NTT^-1
c2 − s^T ∘ c1
m′
G
ss′
KDF
ss
KDF
ss

NTT 적용 영역


sample
e
A
s
NTT
NTT
A ∘ s
sk
A ∘ s + e
pk
polynomial
N = 256
(다항식 최대 차수)
vector
K = 2,3,4
(보안강도)
matrix
K × K
array
ss = 32,
ct = 320K + 128,
pk = 384K + 32,
sk = 384K + pk + 64
Dec
s
c1
c2
NTT
s^T ∘ c1
NTT^-1
c2 − s^T ∘ c1
m′
G
ss′
KDF
ss
ct
pk
sample
m
G
r′
ss′
Enc
e1
A^T
r
p
e2
NTT
A^T ∘ r
p^T ∘ r
NTT^-1
NTT^-1
A^T ∘ r + e1
p^T ∘ r + e2 + m
c1
c2
ct = c1 ‖ c2
KDF
ss

NTT 적용 영역



Uniform sampling in R_q. KYBER uses a deterministic approach to sample elements in R_q that are statistically close to a uniformly random distribution. For this sampling we use a function Parse: $\mathcal{B}^* \to R_q$, which receives as input a byte stream $B = b_0, b_1, b_2, \ldots$ and computes the NTT-representation $\hat{a} = \hat{a}_0 + \hat{a}_1 X + \cdots + \hat{a}_{n-1} X^{n-1} \in R_q$ of $a \in R_q$. Parse is described in Algorithm 1 (note that this description assumes that $q = 3329$).

Algorithm 7 KYBER.CCAKEM.KeyGen()

Output: Public key $pk \in \mathcal{B}^{12 \cdot k \cdot n/8+32}$
Output: Secret key $sk \in \mathcal{B}^{24 \cdot k \cdot n/8+96}$
1: $z \leftarrow \mathcal{B}^{32}$
2: $(pk, sk') := \text{KYBER.CPAPKE.KeyGen}()$
3: $sk := (sk' \| pk \| \text{H}(pk) \| z)$
4: **return** (pk, sk)

Algorithm 4 KYBER.CPAPKE.KeyGen(): key generation

Output: Secret key $sk \in \mathcal{B}^{12 \cdot k \cdot n/8}$
Output: Public key $pk \in \mathcal{B}^{12 \cdot k \cdot n/8+32}$
1: $d \leftarrow \mathcal{B}^{32}$
2: $(\rho, \sigma) := \text{G}(d)$
3: $N := 0$
4: **for** i from 0 to $k-1$ **do**
5: **for** j from 0 to $k-1$ **do**
6: $\hat{\mathbf{A}}[i][j] := \text{Parse}(\text{XOF}(\rho, j, i))$
7: **end for**
8: **end for**
9: **for** i from 0 to $k-1$ **do**
10: $\mathbf{s}[i] := \text{CBD}_{\eta_1}(\text{PRF}(\sigma, N))$
11: $N := N + 1$
12: **end for**
13: **for** i from 0 to $k-1$ **do**
14: $\mathbf{e}[i] := \text{CBD}_{\eta_1}(\text{PRF}(\sigma, N))$
15: $N := N + 1$
16: **end for**
17: $\hat{\mathbf{s}} := \text{NTT}(\mathbf{s})$

Algorithm 1 Parse: $\mathcal{B}^* \to R_q^n$

Input: Byte stream $B = b_0, b_1, b_2, \ldots \in \mathcal{B}^*$
Output: NTT-representation $\hat{a} \in R_q$ of $a \in R_q$
$i := 0$
$j := 0$
while $j < n$ **do**
$d_1 := b_i + 256 \cdot (b_{i+1} \bmod^+ 16)$
$d_2 := \lfloor b_{i+1}/16 \rfloor + 16 \cdot b_{i+2}$
if $d_1 < q$ **then**
$\hat{a}_j := d_1$
$j := j + 1$
end if
if $d_2 < q$ **and** $j < n$ **then**
$\hat{a}_j := d_2$
$j := j + 1$
end if
$i := i + 3$
end while
return $\hat{a}_0 + \hat{a}_1 X + \cdots + \hat{a}_{n-1} X^{n-1}$

암호 알고리즘의 첫 단계인
공개키 행렬 A를 생성할 때부터
NTT 도메인을 전제함

NTT를 사용하지 않기 위해서는
같은 분포를 따르는 sampling 알고리즘을
자체적으로 설계해야 함
(test vector 없음)

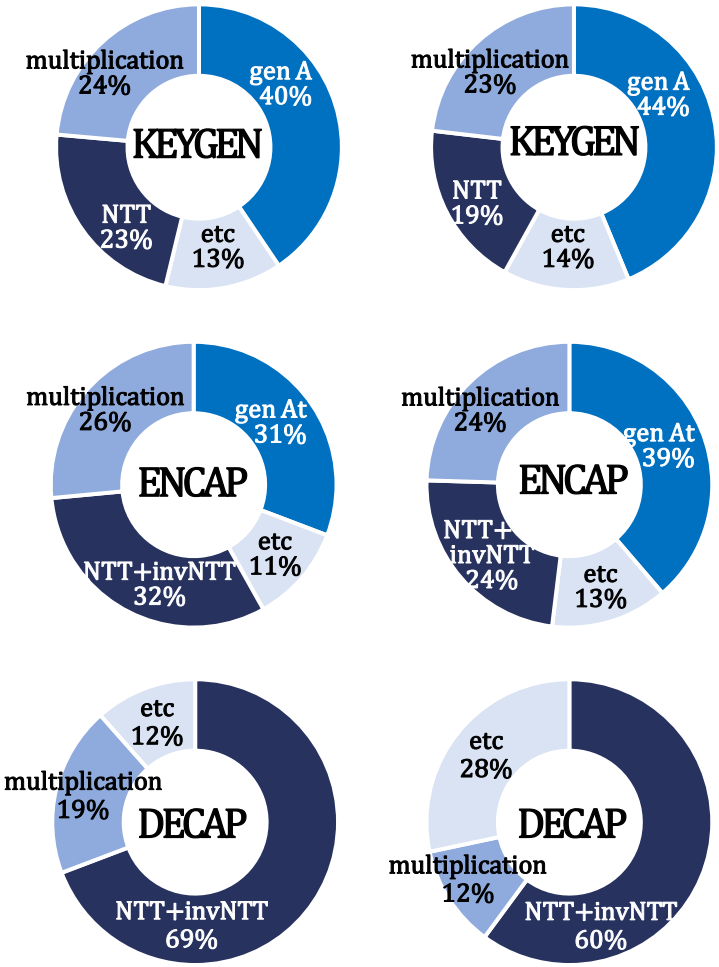
The NTT domain. Computing the discrete Fourier transform on elements from R_q can be done with methods analogous to the fast Fourier transform [31], except that operations on coefficients are defined in a finite field [64]. This is often referred to as the number theoretic transform (NTT). Before being able to define the expansion of the seed ρ into the matrix $\mathbf{A}$, we need to define the NTT domain of polynomials. Let $\omega = 3844 \in \mathbb{Z}_q$ and $\psi = \sqrt{\omega} = 62$, where ψ is chosen as the smallest element of multiplicative order 2^9 in $\mathbb{F}_q^* = \mathbb{F}_{7681}^*$.

Generation of A. Generation of the matrix $\mathbf{A} = (a_{i,j}) \in R_q^{k \times k}$ receives as input the public seed ρ. To generate the entry $a_{i,j} \in R_q$ we first expand ρ through cSHAKE-128 with the 2-byte domain separator (i, j). The output of this expansion is considered a stream of 16-bit little-endian integers. On this sequence of 16-bit integers we run rejection sampling to obtain coefficients in $\{0, \ldots, q-1\}$. The resulting polynomial $a_{i,j}$ is assumed to be in NTT domain.

출처) R. Avanzi, et. al., "CRYSTALS-Kyber Algorithm Specifications And Supporting Documentation (version 3.0)", In the third round of the NIST PQC standardization, 2020.
J. Bos, et al. "CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM." *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2018.

함수별 소요 시간 측정 결과

PC에서 KEM 알고리즘별 세부 함수 동작 속도 비율



보드에서 KEM 알고리즘별 세부 함수 동작 속도 비율

(단위 : ms)

KEM	PC	보드	PKE	PC	보드	세부 함수	PC	보드
KeyGen	0.121	9.12	KeyGen	0.106	8.25	gen A	0.043	3.61
						NTT	0.024	1.55
						multiplication	0.025	1.91
						etc	0.014	1.18
			etc	0.015	0.87			
Encap-sulation	0.143	11.12	Encrypt	0.119	9.30	gen A^T	0.036	3.60
						NTT+invNTT	0.037	2.19
						multiplication	0.031	2.28
						etc	0.015	1.23
			Etc	0.024	1.82			
Decap-sulation	0.163	11.96	Decrypt	0.026	1.73	NTT+invNTT	0.018	1.04
						multiplication	0.005	0.20
						etc	0.003	0.49
			Encrypt	0.119	9.30			
			etc	0.028	0.93			

측정 환경 : PC) 13th Gen Intel(R) Core(TM) i9-13900K 3.00GHz, 128GB RAM
보드) Arty Z7 Xilinx FPGA 650MHz dual-core Cortex-A9 512MB DDR3

- 전체적으로 보드가 PC보다 약 80배정도 느림
- 함수에 따른 병목현상은 PC와 보드가 비슷하게 나타남

KYBER는 다항식 환 $R_q = \mathbb{Z}_q[x]/\langle x^N + 1\rangle$와
$N = 256, = 2^8, q = 3329$를 사용하는 LWE 문제에 기반함

NTT(Number Theoretic Transformation)

다항식 환 $R^+ \coloneqq \mathbb{Z}[x]/\langle x^N + 1\rangle$의 원소를 $R_q^- \coloneqq \mathbb{Z}_q[x]/\langle x^N - 1\rangle$의 원소로 변환한 후
DFT을 적용하는 알고리즘

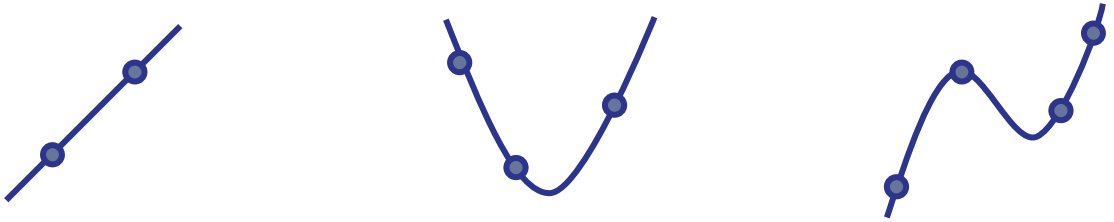
DFT(Discrete Fourier Transformation)

$R_q^- \coloneqq \mathbb{Z}_q[x]/\langle x^N - 1\rangle$의 원소를 R_q^-의 원소로 변환하는 알고리즘

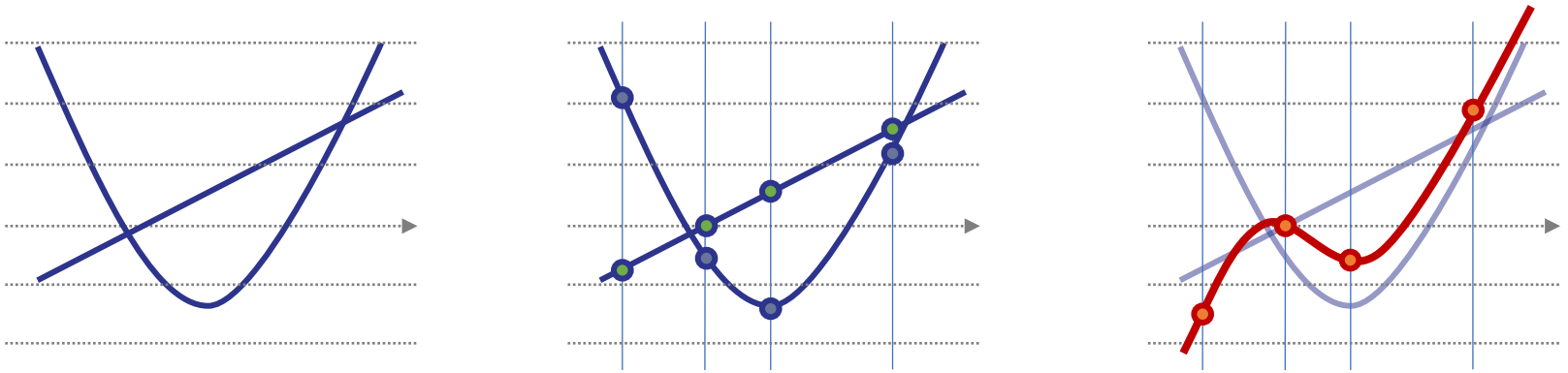
Fast Fourier Transformation(FFT)

$N = 2^k$ $(k \in \mathbb{Z}^+)$일 때, N차 다항식에 대하여 DFT를 $O(N \log N)$으로 계산하는 알고리즘

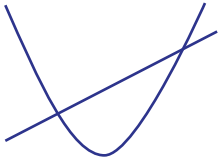
서로 다른 N개의 point로 $N-1$차 다항식을 표현할 수 있다는 아이디어를 통해
다항식 곱셈을 수행할 수 있음 (예를 들어, 1차 다항식은 2개의 point, 2차 다항식은 3개의 point로 표현할 수 있음)



다항식 $A(x), B(x)$이 주어지고, $C(x) = A(x)B(x)$라고 할 때,
$x = \tilde{x}$에 대하여 $C(\tilde{x}) = A(\tilde{x})B(\tilde{x})$이므로 N개의 x_j $(0 \le j < N)$에 대한 $A(x_j), B(x_j)$를 구하면
N개의 point $\left(x_j, C(x_j) = A(x_j)B(x_j)\right)$를 얻어 $C(x)$를 구할 수 있음



이때 $N-1$차 다항식으로부터 N개의 point를 얻는 알고리즘이 FFT이고,
FFT 역연산 알고리즘인 FFT^{-1}로 N개의 point로부터 $N-1$차 다항식을 얻을 수 있음
(*다항식 환* $R_q = \mathbb{Z}_q[x]/\langle x^N+1\rangle$ *위에서의 다항식 곱셈이기 때문*)



$A(x) = a_0 + a_1x + \cdots + a_{N-1}x^{N-1}$
$B(x) = b_0 + b_1x + \cdots + b_{N-1}x^{N-1}$

일반적인 다항식 곱셈
$O(N^2)$

$C(x) = A(x)B(x)$
$= c_0 + c_1x + \cdots + c_{N-1}x^{N-1}$

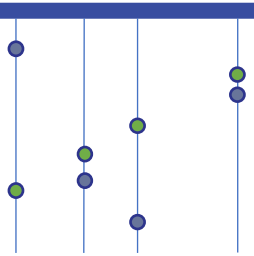
FFT
$O(N \log N)$

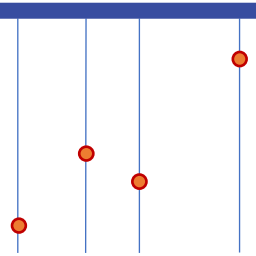
FFT^{-1}
$O(N \log N)$

$\{(x_0, A(x_0)), \ldots, (x_{N-1}, A(x_{N-1}))\} \in P^*(A(x))$
$\{(x_0, B(x_0)), \ldots, (x_{N-1}, B(x_{N-1}))\} \in P^*(B(x))$

Point-wise 곱셈
$O(N)$

$\{(x_0, C(x_0)), \ldots, (x_{N-1}, C(x_{N-1}))\} \in P^*(C(x))$
$with\ C(x_i) = A(x_j)B(x_j)\ for\ 0 \le j \le N-1$





다항식 환 위에서 FFT를 이용한 다항식 곱셈

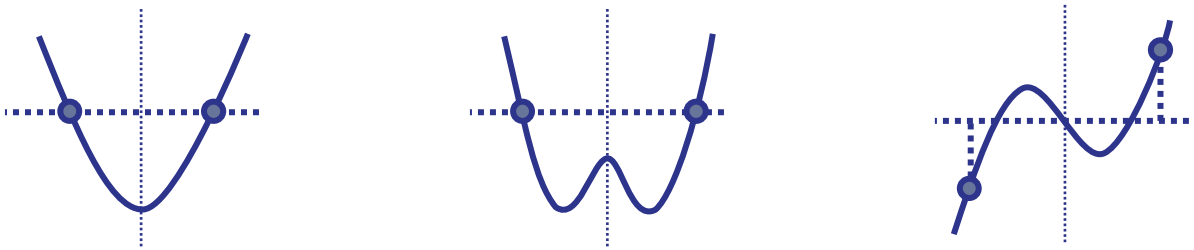
FFT 알고리즘을 설계하는 과정의 핵심은 어떻게 point를 선택할 것인가임

하나의 x에서 얻은 $A(x), B(x)$의 값을 반복해서 사용할 수 있도록 point를 선택함

임의의 다항식 $P(x)$에 대하여

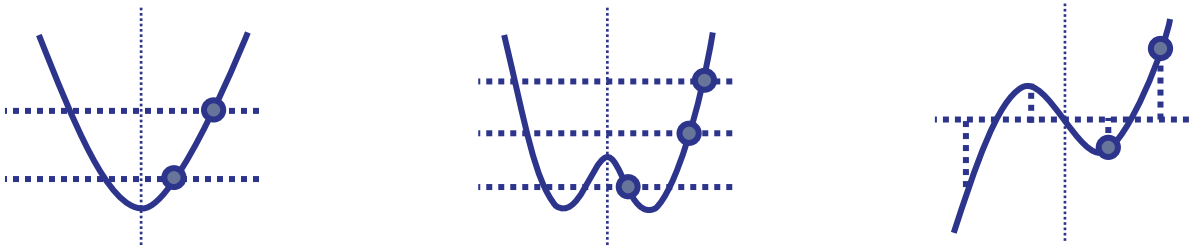
$P(x)$가 짝수 차수 함수인 경우 $(x, P(x))$를 구하면 $(-x, P(x))$가 또다른 point가 되고,

$P(x)$가 홀수 차수 함수인 경우 $(x, P(x))$를 구하면 $(-x, -P(x))$가 또다른 point가 됨



즉, $N-1$차 다항식을 얻기 위하여 N개의 point를 구하는 대신 $N/2$개의 point만 구해도 됨

KYBER 파라미터를 고려하여 이후 나오는 N은 $2^t (t \in \mathbb{Z}^+)$라고 가정



임의의 다항식 $P(x)$의 짝수 차수 항들을 모은 다항식을 $P_e(x)$라고 하고,
홀수 차수 항들을 모은 다항식을 $P_o'(x)$라고 하면

$$P(x) = P_e(x) + P_o'(x)$$

그리고 $P_o'(x)$를 x로 묶으면 $P(x) = P_e(x) + xP_o(x)$를 얻을 수 있음

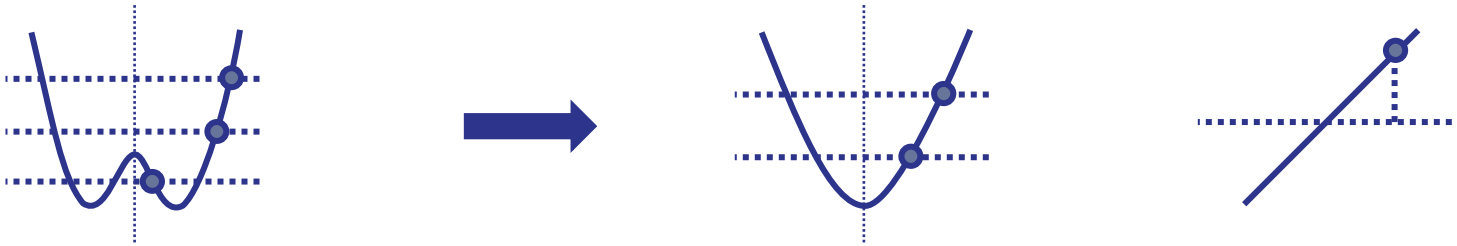
이때 $P(x)$는 다음 성질을 만족함

$$P(x) = P_e(x) + xP_o(x)$$
$$P(-x) = P_e(x) - xP_o(x)$$

$P_e(x)$와 $P_o(x)$는 짝수 차수 항들로만 이루어져 있으므로
x^2을 변수로 갖는 다항식 $P_e(x^2), P_o(x^2)$으로 생각할 수 있음

변수를 치환하면 차수가 각각 $N-1$에서 $\frac{N}{2}-1$로 줄어듦

$P_e(x^2), P_o(x^2)$을 구하기 위해서는 각각 $N/2$개의 서로 다른 point를 찾으면 되고,
$P(x)$는 $N/2 + N/2$개의 point로 구할 수 있음($N/2$개의 point는 따로 구하지 않고 이전에 구한 point를 통해 얻음)



$$P(x) = x^4 + ax^3 + bx^2 + cx + d$$
$$= (x^4 + bx^2 + d) + x(ax^2 + c)$$
$$= P_e(x) + xP_o(x)$$

$$P_e(x) = x^4 + bx^2 + d$$
$$P_e(x^2) = (x^2)^2 + b(x^2) + d$$

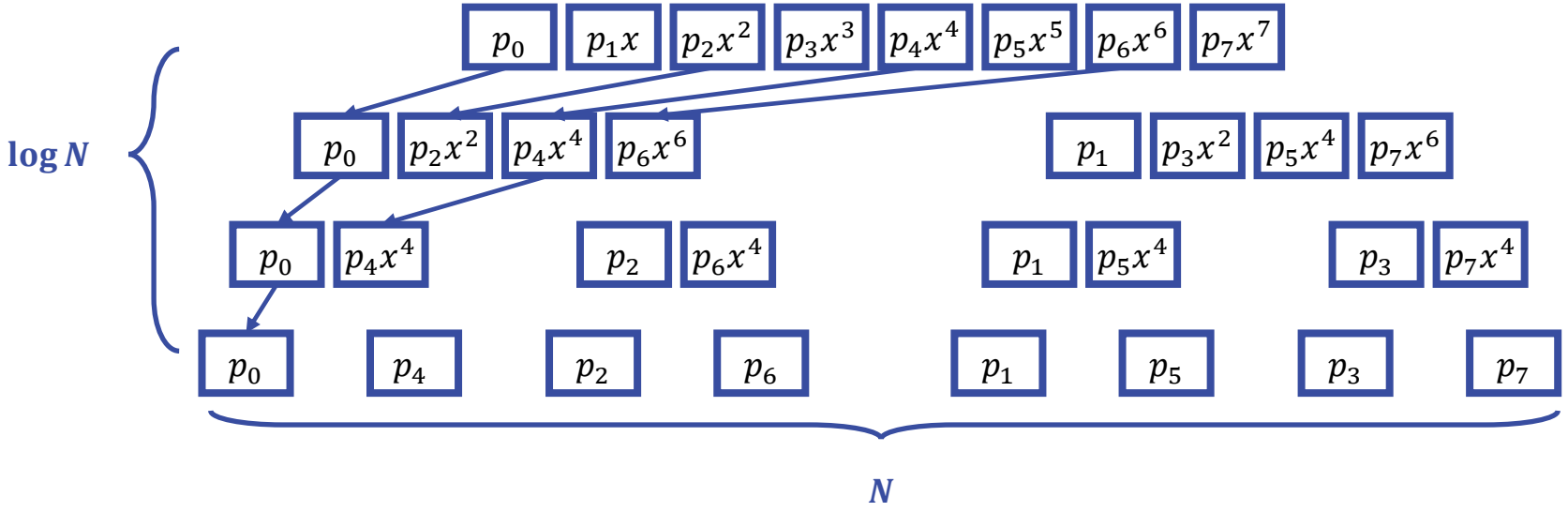
$$P_o(x) = ax^2 + c$$
$$P_o(x^2) = a(x^2) + c$$

$$t = x^2$$

$$P_e(t) = t^2 + bt + d$$
$$= (t^2 + d) + t(b)$$
$$= P_e'(t) + tP_o'(t)$$

$$P_o(t) = at + c$$
$$= (c) + t(a)$$
$$= P_e''(t) + tP_o''(t)$$

$$P(x) = p_0 + p_1 x + \cdots + p_7 x^7$$
$$= (p_0 + p_2 x^2 + p_4 x^4 + p_6 x^6) + x(p_1 + p_3 x^2 + p_5 x^4 + p_7 x^6) = P_e(x^2) + xP_o(x^2)$$



각 단계에서 $N/2$개의 point는 따로 구하지 않고 이전에 구한 point를 통해 얻음

$$P(x) = p_0 + p_1 x + \cdots + p_{N-1}x^{N-1}$$
$$= (p_0 + p_2 x^2 + \cdots + p_{N-2}x^{N-2}) + x(p_1 + p_3 x^2 + \cdots + p_{N-1}x^{N-2}) = P_e(x^2) + xP_o(x^2)$$

$$P(x) = P_e(x^2) + xP_o(x^2)$$

$$P(-x) = P_e(x^2) - xP_o(x^2)$$

$$\left\{(x_0, P(x_0)), (-x_0, P(-x_0)), \ldots, \left(x_{N/2-1}, P\left(x_{N/2-1}\right)\right), \left(-x_{N/2-1}, P(-x_{N/2-1})\right)\right\} \in P^*\left(P(x)\right)$$

$$\left\{(x_0, P_e(x_0^2)), (-x_0, P_e(x_0^2)), (x_1, P_e(x_1^2)), \ldots, \left(x_{N/2-1}, P_e\left(x_{N/2-1}^2\right)\right), \left(-x_{N/2-1}, P_e(x_{N/2-1}^2)\right)\right\} \in P^*\left(P_e(x^2)\right)$$

$$\left\{(x_0, P_o(x_0^2)), (-x_0, P_o(x_0^2)), (x_1, P_o(x_1^2)), \ldots, \left(x_{N/2-1}, P_o\left(x_{N/2-1}^2\right)\right), \left(-x_{N/2-1}, P_o(x_{N/2-1}^2)\right)\right\} \in P^*\left(P_o(x^2)\right)$$

모든 재귀적 과정에서 $P(x)$에 대한 ($P_e\left(x_j^2\right), P_o\left(x_j^2\right)$에 대한) $\pm x_j$ pair는

복소수 상에서 언제나 보장할 수 있음

Point 선택

$\omega := e^{2\pi i/N}$이라고 하면

(Euler's formula $e^{ix} = \cos x + i \sin x$)

$\omega^j = e^{2\pi ij/N}$ $(0 \le j \le N-1)$이고

$-\omega^j = \omega^{j+N/2}$이므로

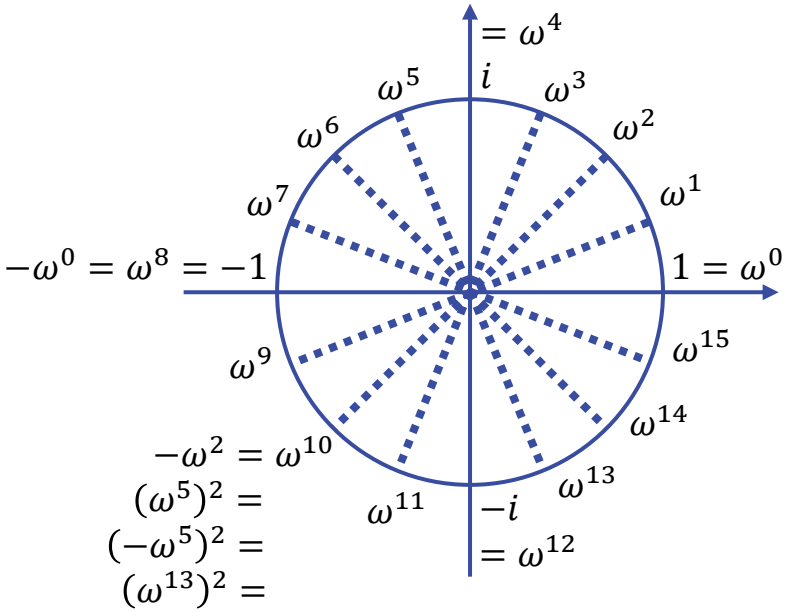
ω^j와 $-\omega^{j+N/2}$는 $\pm x_j$ pair임

ω^j에 대하여 $\omega^{2j}, \omega^{2j+N/2}$으로

쉽게 새로운 $\pm x_j$ pair를 구할 수 있음

⇒ 재귀적 함수를 구현하기에 편리함

($N = 2^t$ 형태이므로 $N/2$를 수행할 때 문제가 없음)



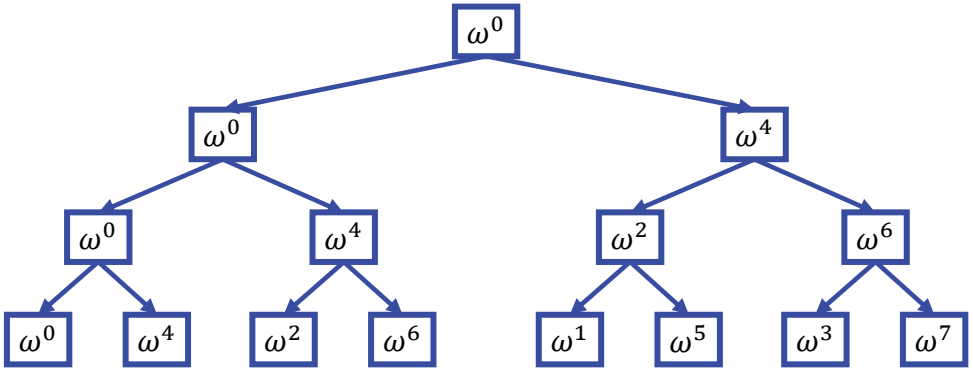
$$P(\omega) = p_0 + p_1\omega + \cdots + p_{N-1}\omega^{N-1}$$
$$= (p_0 + p_2\omega^2 + \cdots + p_{N-2}\omega^{N-2}) + \omega(p_1 + p_3\omega^2 + \cdots + p_{N-1}\omega^{N-2}) = P_e(\omega^2) + \omega P_o(\omega^2)$$

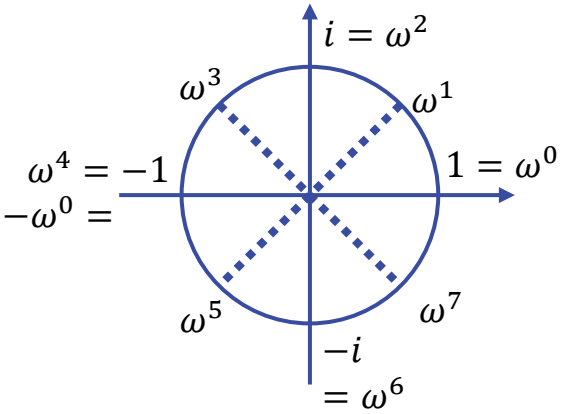
$$P(\omega) = P_e(\omega^2) + \omega P_o(\omega^2)$$

$$P(-\omega) = P_e(\omega^2) - \omega P_o(\omega^2)$$

$$\left\{(\omega^0, P(\omega^0)), (-\omega^0, P(-\omega^0)), \ldots, \left(\omega^{N/2-1}, P\left(\omega^{N/2-1}\right)\right), \left(-\omega^{N/2-1}, P(-\omega^{N/2-1})\right)\right\} \in P^*\left(P(x)\right)$$

$$\Leftrightarrow \left\{(\omega^0, P(\omega^0)), \left(\omega^{N/2}, P(\omega^{N/2})\right), \ldots, \left(\omega^{N/2-1}, P\left(\omega^{N/2-1}\right)\right), (\omega^{N-1}, P(\omega^{N-1}))\right\} \in P^*\left(P(x)\right)$$





$n = 8, w \leftarrow e^{\frac{2\pi i}{8}} = \omega$

$P(x) = [\ p_0\ \ p_1\ \ p_2\ \ p_3\ \ p_4\ \ p_5\ \ p_6\ \ p_7\ ]$ → $\mathrm{NTT}(P(x)) = [\ c_0\ \ c_1\ \ c_2\ \ c_3\ \ c_4\ \ c_5\ \ c_6\ \ c_7\ ]$

$P_e = [\ p_0\ \ p_2\ \ p_4\ \ p_6\ ]$

$n = 4\ \ w \leftarrow e^{\frac{2\pi i}{4}} = \omega^2$

$P_e = [\ p_0\ \ p_4\ ]$

$n = 2, w \leftarrow e^{\frac{2\pi i}{2}} = \omega^4$

$y_e = p_0 \qquad y_o = p_4$

$\boldsymbol{y[0]} = y_e[0] + w^0 y_o[0] = \boldsymbol{p_0 + \omega^0 p_4}$

$\boldsymbol{y[1]} = y_e[0] - w^0 y_o[0] = \boldsymbol{p_0 - \omega^0 p_4}$

$y_e = [\ y[0]\ \ y[1]\ ]$

$P_o = [\ p_2\ \ p_6\ ]$

$n = 2, w \leftarrow e^{\frac{2\pi i}{2}} = \omega^4$

$y_e = p_2 \qquad y_o = p_6$

$\boldsymbol{y[0]} = y_e[0] + w^0 y_o[0] = \boldsymbol{p_2 + \omega^0 p_6}$

$\boldsymbol{y[1]} = y_e[0] - w^0 y_o[0] = \boldsymbol{p_2 - \omega^0 p_6}$

$y_o = [\ y[0]\ \ y[1]\ ]$

$\boldsymbol{y[0]} = y_e[0] + w^0 y_o[0] = (\boldsymbol{p_0 + \omega^0 p_4}) + \boldsymbol{\omega^0}(\boldsymbol{p_2 + \omega^0 p_6})$

$\boldsymbol{y[2]} = y_e[0] - w^0 y_o[0] = (\boldsymbol{p_0 + \omega^0 p_4}) - \boldsymbol{\omega^0}(\boldsymbol{p_2 + \omega^0 p_6})$

$\boldsymbol{y[1]} = y_e[1] + w^1 y_o[1] = (\boldsymbol{p_0 - \omega^0 p_4}) + \boldsymbol{\omega^2}(\boldsymbol{p_2 - \omega^0 p_6})$

$\boldsymbol{y[3]} = y_e[1] - w^1 y_o[1] = (\boldsymbol{p_0 - \omega^0 p_4}) - \boldsymbol{\omega^2}(\boldsymbol{p_2 - \omega^0 p_6})$

$y_e = [\ y[0]\ \ y[1]\ y[2]\ y[3]\ ]$

$P_o = [\ p_1\ \ p_3\ \ p_5\ \ p_7\ ]$

$n = 4\ \ w \leftarrow e^{\frac{2\pi i}{4}} = \omega^2$

$P_e = [\ p_1\ \ p_5\ ]$

$n = 2, w \leftarrow e^{\frac{2\pi i}{2}} = \omega^4$

$y_e = p_1 \qquad y_o = p_5$

$\boldsymbol{y[0]} = y_e[0] + w^0 y_o[0] = \boldsymbol{p_1 + \omega^0 p_5}$

$\boldsymbol{y[1]} = y_e[0] - w^0 y_o[0] = \boldsymbol{p_1 - \omega^0 p_5}$

$y_e = [\ y[0]\ \ y[1]\ ]$

$P_o = [\ p_3\ \ p_7\ ]$

$n = 2, w \leftarrow e^{\frac{2\pi i}{2}} = \omega^4$

$y_e = p_3 \qquad y_o = p_7$

$\boldsymbol{y[0]} = y_e[0] + w^0 y_o[0] = \boldsymbol{p_3 + \omega^0 p_7}$

$\boldsymbol{y[1]} = y_e[0] - w^0 y_o[0] = \boldsymbol{p_3 - \omega^0 p_7}$

$y_o = [\ y[0]\ \ y[1]\ ]$

$\boldsymbol{y[0]} = y_e[0] + w^0 y_o[0] = (\boldsymbol{p_1 + \omega^0 p_5}) + \boldsymbol{\omega^0}(\boldsymbol{p_3 + \omega^0 p_7})$

$\boldsymbol{y[2]} = y_e[0] - w^0 y_o[0] = (\boldsymbol{p_1 + \omega^0 p_5}) - \boldsymbol{\omega^0}(\boldsymbol{p_3 + \omega^0 p_7})$

$\boldsymbol{y[1]} = y_e[1] + w^1 y_o[1] = (\boldsymbol{p_1 - \omega^0 p_5}) + \boldsymbol{\omega^2}(\boldsymbol{p_3 - \omega^0 p_7})$

$\boldsymbol{y[3]} = y_e[1] - w^1 y_o[1] = (\boldsymbol{p_1 - \omega^0 p_5}) - \boldsymbol{\omega^2}(\boldsymbol{p_3 - \omega^0 p_7})$

$y_o = [\ y[0]\ \ y[1]\ y[2]\ y[3]\ ]$

$\boldsymbol{c_0} = y[0] = y_e[0] + w^0 y_o[0] = (\boldsymbol{p_0 + \omega^0 p_4}) + \boldsymbol{\omega^0}(\boldsymbol{p_2 + \omega^0 p_6}) + \boldsymbol{\omega^0}\left((\boldsymbol{p_1 + \omega^0 p_5}) + \boldsymbol{\omega^0}(\boldsymbol{p_3 + \omega^0 p_7})\right)$

$\boldsymbol{c_4} = y[4] = y_e[0] - w^0 y_o[0] = (\boldsymbol{p_0 + \omega^0 p_4}) + \boldsymbol{\omega^0}(\boldsymbol{p_2 + \omega^0 p_6}) - \boldsymbol{\omega^0}\left((\boldsymbol{p_1 + \omega^0 p_5}) + \boldsymbol{\omega^0}(\boldsymbol{p_3 + \omega^0 p_7})\right)$

$\boldsymbol{c_1} = y[1] = y_e[1] + w^1 y_o[1] = (\boldsymbol{p_0 - \omega^0 p_4}) + \boldsymbol{\omega^2}(\boldsymbol{p_2 - \omega^0 p_6}) + \boldsymbol{\omega^1}\left((\boldsymbol{p_1 - \omega^0 p_5}) + \boldsymbol{\omega^2}(\boldsymbol{p_3 - \omega^0 p_7})\right)$

$\boldsymbol{c_5} = y[5] = y_e[1] - w^1 y_o[1] = (\boldsymbol{p_0 - \omega^0 p_4}) + \boldsymbol{\omega^2}(\boldsymbol{p_2 - \omega^0 p_6}) - \boldsymbol{\omega^1}\left((\boldsymbol{p_1 - \omega^0 p_5}) + \boldsymbol{\omega^2}(\boldsymbol{p_3 - \omega^0 p_7})\right)$

$\boldsymbol{c_2} = y[2] = y_e[2] + w^2 y_o[2] = (\boldsymbol{p_0 + \omega^0 p_4}) - \boldsymbol{\omega^0}(\boldsymbol{p_2 + \omega^0 p_6}) + \boldsymbol{\omega^2}\left((\boldsymbol{p_1 + \omega^0 p_5}) - \boldsymbol{\omega^0}(\boldsymbol{p_3 + \omega^0 p_7})\right)$

$\boldsymbol{c_6} = y[6] = y_e[2] - w^2 y_o[2] = (\boldsymbol{p_0 + \omega^0 p_4}) - \boldsymbol{\omega^0}(\boldsymbol{p_2 + \omega^0 p_6}) - \boldsymbol{\omega^2}\left((\boldsymbol{p_1 + \omega^0 p_5}) - \boldsymbol{\omega^0}(\boldsymbol{p_3 + \omega^0 p_7})\right)$

$\boldsymbol{c_3} = y[3] = y_e[3] + w^3 y_o[3] = (\boldsymbol{p_0 - \omega^0 p_4}) - \boldsymbol{\omega^2}(\boldsymbol{p_2 - \omega^0 p_6}) + \boldsymbol{\omega^3}\left((\boldsymbol{p_1 - \omega^0 p_5}) - \boldsymbol{\omega^2}(\boldsymbol{p_3 - \omega^0 p_7})\right)$

$\boldsymbol{c_7} = y[7] = y_e[3] - w^3 y_o[3] = (\boldsymbol{p_0 - \omega^0 p_4}) - \boldsymbol{\omega^2}(\boldsymbol{p_2 - \omega^0 p_6}) - \boldsymbol{\omega^3}\left((\boldsymbol{p_1 - \omega^0 p_5}) - \boldsymbol{\omega^2}(\boldsymbol{p_3 - \omega^0 p_7})\right)$

DFT point 구하기

$$P(x) = p_0 + p_1 x + \cdots + p_{N-1}x^{N-1}, \text{ point } x_0, x_1, \ldots, x_{N-1}$$

$$\begin{pmatrix} P(x_0) \\ P(x_1) \\ \vdots \\ P(x_{N-1}) \end{pmatrix} = \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{N-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N-1} & x_{N-1}^2 & \cdots & x_{N-1}^{N-1} \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_{N-1} \end{pmatrix}$$

$$\text{point } \omega^0, \omega^1, \ldots, \omega^{N-1}$$

$$\underbrace{\begin{pmatrix} P(\omega^0) \\ P(\omega^1) \\ \vdots \\ P(\omega^{N-1}) \end{pmatrix}}_{\text{point}} = \underbrace{\begin{pmatrix} 1 & \omega^0 & \omega^{0\cdot 1} & \cdots & \omega^{0(N-1)} \\ 1 & \omega^1 & \omega^{1\cdot 1} & \cdots & \omega^{1(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{(N-1)\cdot 1} & \cdots & \omega^{(N-1)(N-1)} \end{pmatrix}}_{\text{DFT(discrete Fourier transform) matrix}} \underbrace{\begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_{N-1} \end{pmatrix}}_{\text{coefficient}}$$

DFT point 구하기

$$\text{point } \omega^0, \omega^1, \dots, \omega^{N-1}$$

$$\underbrace{\begin{pmatrix} P(\omega^0) \\ P(\omega^1) \\ \vdots \\ P(\omega^{N-1}) \end{pmatrix}}_{\text{point}} = \underbrace{\begin{pmatrix} 1 & \omega^0 & \omega^{0\cdot 1} & \cdots & \omega^{0(N-1)} \\ 1 & \omega^1 & \omega^{1\cdot 1} & \cdots & \omega^{1(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{(N-1)\cdot 1} & \cdots & \omega^{(N-1)(N-1)} \end{pmatrix}}_{\text{DFT(discrete Fourier transform) matrix}} \underbrace{\begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_{N-1} \end{pmatrix}}_{\text{coefficient}}$$

$$\left\{(\omega^0, P(\omega^0)), \left(\omega^{N/2}, P(\omega^{N/2})\right), \dots, \left(\omega^{N/2-1}, P\left(\omega^{N/2-1}\right)\right), (\omega^{N-1}, P(\omega^{N-1}))\right\} \in P^*\left(P(x)\right)$$

KYBER 연산을 수행하는 다항식 환 위에서 ω와 같이 사용할 수 있는 특정 값 $\tilde{x}$가 존재함

이 값들로 $P(\tilde{x}), P(-\tilde{x})$를 계산하며 $P^*\left(P(x)\right)$의 point들을 구하는 알고리즘이 FFT임

NTT(Number Theoretic Transformation)는
다항식 환 $R^+ := \mathbb{Z}[x]/\langle x^N + 1\rangle$의 원소를 $R_q^- := \mathbb{Z}_q[x]/\langle x^N - 1\rangle$의 원소로 변환한 후
FFT을 적용하는 알고리즘

알고리즘 변형을 통하여 recursive FFT를 iteration method로 구현할 수 있음

Algorithm 1 Forward NTT of a polynomial $f = c_0 + c_1 X \ldots c_{n-1} X^{n-1} \in \mathbb{Z}_q[X]/(X^n + 1)$ with precomputed roots of unity $\zeta_k = \zeta^{\mathrm{brv}(k)}$, $0 \le k < n$.

$k \leftarrow 1$
for $l \leftarrow n/2;\ l > 0;\ l \leftarrow l/2$ **do**
  for $s \leftarrow 0;\ s < n;\ s \leftarrow j + l$ **do**
    for $j \leftarrow s;\ j < s + l;\ j \leftarrow j + 1$ **do**
      $t \leftarrow \zeta_k \cdot c_{j+l}$
      $c_{j+l} \leftarrow c_j - t$
      $c_j \leftarrow c_j + t$
    end for
    $k \leftarrow k + 1$
  end for
end for

```
void ntt(int16_t r[256]) {
  unsigned int len, start, j, k;
  int16_t t, zeta;

  k = 1;
  for(len = 128; len >= 2; len >>= 1) {
    for(start = 0; start < 256; start = j + len) {
      zeta = zetas[k++];
      for(j = start; j < start + len; ++j) {

        t = fqmul(zeta, r[j + len]);

        r[j + len] = r[j] - t;

        r[j] = r[j] + t;
      }
    }
  }
}
```

출처) G. Seiler, "Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography." *Cryptology ePrint Archive,* 2018.

Modular multiplication을 사용하는 이유

임의의 큰 수를 계속 곱해나가면 결과 값이 빠르게 증가함
수가 커질수록 모듈러(나눗셈) 연산에 드는 시간이 크게 증가함

→ 매번 modular 연산을 수행하며 곱셈을 수행하는 것이
메모리 사용과 시간 소비 면에서 효율적임

Montgomery Multiplication은
modular 연산과 함께 곱셈을 수행하는
constant-time multiplication임

```
void ntt(int16_t r[256]) {
  unsigned int len, start, j, k;
  int16_t t, zeta;

  k = 1;
  for(len = 128; len >= 2; len >>= 1) {
    for(start = 0; start < 256; start = j + len) {
      zeta = zetas[k++];
      for(j = start; j < start + len; ++j) {

        t = fqmul(zeta, r[j + len]);

        r[j + len] = r[j] - t;

        r[j] = r[j] + t;
      }
    }
  }
}
```

Montgomery multiplication

```
static int16_t fqmul(int16_t a, int16_t b) {
  return montgomery_reduction((int32_t)a*b);
}
```

출처) G. Seiler, "Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography." *Cryptology ePrint Archive,* 2018.

Montgomery Multiplication

Montgomery Representation

$x_R = xR, y_R = yR$

$x_R \cdot y_R = xR \cdot yR = xyR^2$

$(xy)_R = xyR = (xR \cdot yR)/R = x_R y_R / R$

Montgomery Reduction

$$\begin{aligned} x_R &\rightarrow x_R R^{-1} \bmod Q \\ &= x \bmod Q \\ x_R \cdot y_R &\rightarrow x_R y_R R^{-1} \bmod Q \\ &= xyR \bmod Q \\ &= (xy)_R \bmod Q \\ (xy)_R \cdot z_R &\rightarrow (xyz)_R \bmod Q \end{aligned}$$

Algorithm 3 Signed Montgomery reduction

Require: $0 < q < \frac{\beta}{2}$ odd, $-\frac{\beta}{2}q \leq a = a_1\beta + a_0 < \frac{\beta}{2}q$ where $0 \leq a_0 < \beta$
Ensure: $r' \equiv \beta^{-1}a \pmod q$, $-q < r' < q$

1: $m \leftarrow a_0 q^{-1} \bmod^{\pm} \beta$ ▷ signed low product, q^{-1} precomputed
2: $t_1 \leftarrow \left\lfloor \frac{mq}{\beta} \right\rfloor$ ▷ signed high product
3: $r' \leftarrow a_1 - t_1$

```
static int16_t fqmul(int16_t a, int16_t b) {
  return montgomery_reduction((int32_t)a*b);
}
```

For input $a_R \cdot b_R$,
Return $a_R b_R (2^{16})^{-1} \bmod Q$

```
int16_t montgomery_reduction(int32_t a)
{
    int16_t m;
    int32_t t;

    m = a*QINV; /* a_0*q^{-1} mod 2^16;
    t = (a - (int32_t)m*KYBER_Q) >> 16;
    return t;
}
```

출처) G. Seiler, "Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography." *Cryptology ePrint Archive,* 2018.

연구 방향성

- NTT를 사용하지 않는 KYBER 구현
  1. NTT domain에서 sampling하여 얻은 행렬 A를
     일반적인 sampling 공간의 행렬로 변환할 수 있는지 확인
  2. NTT 없이 동작하는 KYBER를 구현할 때의 test vector 생성 및 검증
  3. Modular multiplication 없이 연산할 때 성능 부담이 없는지 확인

감사합니다