KYBER에 적용된 NTT와 Montgomery reduction

231102 류지은

INDEX

2018 IEEE European Symposium on Security and Privacy

CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM

Joppe Bos*, Léo Ducas†, Eike Kiltz‡, Tancrède Lepoint§, Vadim Lyubashevsky¶,
John M. Schanck‖, Peter Schwabe**, Gregor Seiler††, Damien Stehlé‡‡,
*NXP Semiconductors, Belgium. Email: joppe.bos@nxp.com
†CWI Amsterdam, The Netherlands. Email: ducas@cwi.nl
‡Ruhr-University Bochum, Germany. Email: eike.kiltz@ruh.de
§SRI International, USA. Email: tancrede.lepoint@sri.com
¶IBM Research Zurich, Switzerland. Email: vad@zurich.ibm.com
‖University of Waterloo, Canada. Email: jschanck@uwaterloo.ca
**Radboud University, The Netherlands. Email: peter@cryptojedi.org
††IBM Research Zurich, Switzerland. Email: grs@zurich.ibm.com
‡‡ENS de Lyon, France. Email: damien.stehle@ens-lyon.fr

***Abstract*—Rapid advances in quantum computing, together with the announcement by the National Institute of Standards and Technology (NIST) to define new standards for digital-signature, encryption, and key-establishment protocols, have created significant interest in post-quantum cryptographic schemes.**

This paper introduces Kyber **(part of CRYSTALS – *Cryptographic Suite for Algebraic Lattices* – a package submitted to NIST post-quantum standardization effort in November 2017), a portfolio of post-quantum cryptographic primitives built around a key-encapsulation mechanism (KEM), based on hardness assumptions over module lattices. Our KEM is most naturally seen as a successor to the** NewHope **KEM (Usenix 2016). In particular, the key and ciphertext sizes of our new construction are about half the size, the KEM offers CCA instead of only passive security, the security is based on a more general (and flexible) lattice problem, and our optimized implementation results in essentially the same running time as the aforementioned scheme.**

We first introduce a CPA-secure public-key encryption scheme, apply a variant of the Fujisaki-Okamoto transform to create a CCA-secure KEM, and eventually construct, in a black-box manner, CCA-secure encryption, key exchange, and authenticated-key-exchange schemes. The security of our primitives is based on the hardness of Module-LWE in the classical and quantum random oracle models, and our concrete parameters conservatively target more than 128 bits of post-quantum security.

1. Introduction

There has been an increased interest in post-quantum cryptographic schemes triggered by recent advances in quantum computing [35] and the announcement by the National Institute of Standards and Technology (NIST) to define new standards for digital-signature, encryption, and key-establishment protocols [28]. Constructions based on the hardness of lattice problems are considered to be one of the leading candidates to replace the currently used schemes based on the believed hardness of the traditional number-theoretic problems such as integer factorization and discrete logarithms.

Lattice cryptography initially gained a lot of interest in the theoretical community due to the fact that the designs for cryptographic constructions were accompanied by security proofs based on *worst-case* instances of lattice problems. The first lattice-based encryption scheme was proposed by Ajtai and Dwork [1]. This scheme was later simplified and improved upon by Regev in [67], [68]. One of the major achievements of Regev's work was the introduction of an intermediate problem – the Learning With Errors (LWE) Problem – which was relatively simple to use in cryptographic constructions and asymptotically at least as hard as some standard worst-case lattice problems [61], [25].

The LWE assumption states that it is hard to distinguish from uniform the distribution $(\mathbf{A}, \mathbf{As} + \mathbf{e})$, where $\mathbf{A}$ is a uniformly-random matrix in $\mathbb{Z}_q^{m \times n}$, $\mathbf{s}$ is a uniformly-random vector in $\mathbb{Z}_q^n$, and $\mathbf{e}$ is a vector with random "small" coefficients chosen from some distribution. Applebaum et al. [6] showed that the secret $\mathbf{s}$ in the LWE problem does not need to be chosen uniformly at random: the problem remains hard if $\mathbf{s}$ is chosen from the same narrow distribution as the errors $\mathbf{e}$. Based on the idea from the NTRU cryptosystem [43] of working with elements over polynomial rings rather than over the integers, and following a series of works on this topic [58], [56], [63], [71], Lyubashevsky et al. [57] showed that it is also hard to distinguish a variant of the LWE distribution from the uniform one over certain polynomial rings, thus defining the Ring-LWE assumption.

The combination of all of the above results finally led

DOI 10.1109/EuroSP.2018.00032

Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography

Gregor Seiler*

IBM Research Zurich
grs@zurich.ibm.com

Abstract. Constant-time polynomial multiplication is one of the most time-consuming operations in many lattice-based cryptographic constructions. For schemes based on the hardness of Ring-LWE in power-of-two cyclotomic fields with completely splitting primes, the AVX2 optimized implementation of the Number-Theoretic Transform (NTT) from the NewHope key-exchange scheme is the state of the art for fast multiplication. It uses floating point vector instructions. We show that by using a modification of the Montgomery reduction algorithm that enables a fast approach with integer instructions, we can improve on the polynomial multiplication speeds of NewHope and Kyber by a factor of 4.2 and 6.3 on Skylake, respectively.

Keywords: lattice cryptography · NTT · implementation · AVX

1 Introduction

Lattice-based cryptography has emerged as a promising candidate for public-key cryptography that is still secure after the likely advent of quantum computers.

From a computational point of view, many lattice-based cryptographic schemes are based on operations in polynomial rings of the form $(\mathbb{Z}/q\mathbb{Z})[X]/\langle f(x)\rangle$ where f is an irreducible polynomial over $\mathbb{Z}$ and q is a prime number. For schemes whose security is based on the (presumed) hardness of the Ring-LWE problem, f is usually chosen to be a power-of-two cyclotomic $X^n + 1$ whose roots have order $2n$ and q is a prime that splits completely (in the number field). The latter condition is equivalent to $q \equiv 1 \pmod{2n}$. The reason for power-of-two cyclotomic rings and fully splitting primes is that the splitting behaviour of such primes in these rings allows for fast multiplication using the Fast Fourier Transform (FFT), which is also called the Number-Theoretic Transform (NTT) if it is performed over the base field $\mathbb{Z}_q$. A full NTT-based multiplication of two polynomials needs two forward NTTs to transform the input polynomials, a cheap pointwise vector multiplication and one inverse NTT. An advantage of NTT-based multiplication over other multiplication methods is that one can often save NTTs by sampling polynomials directly in the NTT domain, by storing the NTT domain representation of polynomials for later use, or by making use of the linearity of the NTT when computing sums of products of polynomials. This leads to important speed-ups as multiplication is frequently the most time consuming single operation in lattice-based schemes.

Producing fast constant-time implementations of the NTT for power-of-two cyclotomic fields has received considerable attention in cryptography during the last couple of years [GOPS13, ADPS16, GS16b, LN16]. For optimal speed on current Intel processors one has to exploit the fact that the NTT is easily vectorizable and needs to make use of

*Gregor Seiler was supported by the SNSF ERC Transfer Starting Grant CRETP2-166734-FELICITY and the H2020 Project SAFEcrypto

CRYSTALS-Kyber

Algorithm Specifications And Supporting Documentation
(version 3.0)

Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint,
Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, Damien Stehlé

October 1, 2020

KYBER

격자 기반 KEM(key encapsulation mechanism)

Lattice

Basis vector로 생성 가능한 point들에 의해 주기적인 pattern을 이루는 grid

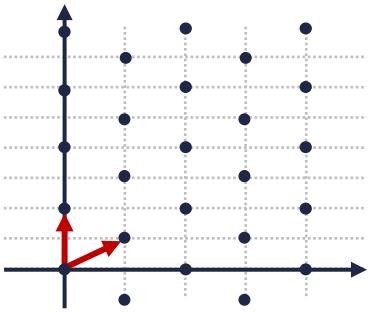
→ 서로 다른 basis가 하나의 같은 lattic를 만들어낼 수 있음

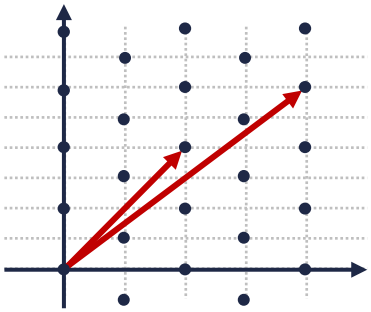
SVP(shortest vector problem)

주어진 basis에 대하여 가장 짧은 non-zero vector를 찾는 문제

⇔ 원점과 가장 가까운 point를 찾는 문제

Ajtai에 의해 NP 문제임이 증명됨





CVP(closest vector problem)

어떤 (grid 위 point가 아닐 수도 있는) point와 가장 가까운 point를 찾는 문제

Ex1. Basis 사이 각이 orthogonal에 가까운 경우

격자 $\mathcal{L} = \{z_1\boldsymbol{b}_1 + z_2\boldsymbol{b}_2\} = \left\{z_1\begin{bmatrix}5\\1\end{bmatrix} + z_2\begin{bmatrix}-2\\8\end{bmatrix}\right\}$에 대하여$(\theta\approx 87)$ point $\begin{bmatrix}27\\8\end{bmatrix}$가 주어졌을 때,

$\begin{cases}5z_1 - 2z_2 = 27\\1z_1 + 8z_2 = 8\end{cases} \Rightarrow \begin{cases}z_1 = 5.52\\z_2 = 0.309\end{cases}$ 이고, 정수로 반올림하면 $(z_1, z_2) = (6, 0)$

이를 대입하여 격자 위 point를 찾으면 $\begin{bmatrix}5 & -2\\1 & 8\end{bmatrix}\begin{bmatrix}6\\0\end{bmatrix} = \begin{bmatrix}30\\6\end{bmatrix}$

Ex2. Basis 사이 각이 orthogonal과 먼 경우

격자 $\mathcal{L} = \{z_1\boldsymbol{b}_1 + z_2\boldsymbol{b}_2\} = \left\{z_1\begin{bmatrix}37\\41\end{bmatrix} + z_2\begin{bmatrix}103\\113\end{bmatrix}\right\}$에 대하여$(\theta\approx 0)$ point $\begin{bmatrix}27\\8\end{bmatrix}$가 주어졌을 때,

$\begin{cases}37z_1 + 103z_2 = 27\\41z_1 + 113z_2 = 8\end{cases} \Rightarrow \begin{cases}z_1 = -53.023\\z_2 = 19.309\end{cases}$ 이고, 정수로 반올림하면 $(z_1, z_2) = (-53, 19)$

이를 대입하여 격자 위 point를 찾으면 $\begin{bmatrix}37 & 103\\41 & 113\end{bmatrix}\begin{bmatrix}-53\\19\end{bmatrix} = \begin{bmatrix}-4\\-26\end{bmatrix}$

하지만 가장 가까운 점은 $(z_1, z_2) = (-52, 19)$일 때의 point $\begin{bmatrix}33\\15\end{bmatrix}$

LWE(learning with Error)

Regev에 의해 LWE 문제가 SVP 문제와 같은 수준의 어려움을 가짐이 증명됨

Learning without error

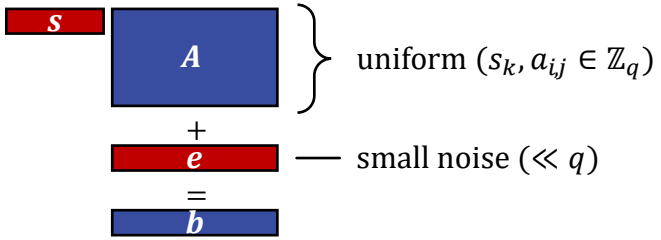
modulus $q \in \mathbb{Z}_{>0}$, dimension $n > 0$이고
$m > 0$인 sample의 개수 m에 대하여,
m개의 vector $a_i \in \mathbb{Z}_q^n$와
values $\boldsymbol{b}_1 = \langle \boldsymbol{s}, \boldsymbol{a}_1 \rangle, \ldots, \boldsymbol{b}_m = \langle \boldsymbol{s}, \boldsymbol{a}_m \rangle$를 알 때,
$\boldsymbol{s} \in \mathbb{Z}_q^n$를 찾는 문제

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

$$pk = \begin{cases} c_{11}x + c_{12}y + c_{13}z + c_{14}w = c_{15} \\ \vdots \\ c_{n1}x + c_{n2}y + c_{n3}z + c_{n4}w = c_{n5} \end{cases}$$

pk가 주어지면 가우스 소거법을 활용하여
쉽게 sk를 구할 수 있음

Learning with error



Break the problem
1. $(\boldsymbol{A}, \boldsymbol{b}) \Rightarrow \boldsymbol{s}$
2. $(\boldsymbol{A}, \boldsymbol{b}) =? (\boldsymbol{A}, \boldsymbol{u})$
   └ uniform ($u_i \in \mathbb{Z}_q$)

$$pk + e$$

$$= \begin{cases} c_{11}x + c_{12}y + c_{13}z + c_{14}w + e_1 = c_{15} + e_1 \\ \vdots \\ c_{n1}x + c_{n2}y + c_{n3}z + c_{n4}w + e_n = c_{n5} + e_n \end{cases}$$

sk를 구하기 어려움

PKE Cryptosystem (KYBER)

Alice

Bob

Key generation

sk

A s $+$ e $=$ t

$n\times1$ $m\times n$ $m\times1$ $m\times1$

pk

Encryption

r $1\times m$

$A \| t$ $m\times(n+1)$

$+$ $e_1 \| e_2$

$+$ $0 \| m$

$=$ $u \| v$ $1\times(n+1)$

Decryption

$v - u \; s = m + e'$

$1\times n$ $n\times1$

무시할 수 있는
작은 noise 값

PKE Cryptosystem

Alice

$R = \mathbb{Z}_q[X](X^n + 1), \chi \leftarrow R$

Bob

Key generation

Generate $\boldsymbol{A} \in R^{k \times k}$,

$\boldsymbol{s}, \boldsymbol{e} \leftarrow \chi^k + \boldsymbol{e} = \boldsymbol{t}$

$\boldsymbol{As} + \boldsymbol{e} = \boldsymbol{t}$

pk : $\boldsymbol{A}, \boldsymbol{t}$

sk : $\boldsymbol{s}$ ($\boldsymbol{e}$: secret)

Send $(\boldsymbol{A}, \boldsymbol{t})$

Encryption

Generate $m \in \{0,1\}^n$,

$\boldsymbol{r}, \boldsymbol{e}_1 \leftarrow \chi^k, e_2 \leftarrow \chi$

$\boldsymbol{u} = \boldsymbol{rA} + \boldsymbol{e}_1$,

$v = \boldsymbol{rt} + e_2 + m$

Save m

Send $(\boldsymbol{u}, v)$

Decryption

$v - \boldsymbol{us} = m + e' \approx m$

Save m

How to decrypt the message m

$$
\begin{aligned}
v &= \boldsymbol{rt} + e_2 + m = \boldsymbol{r}(\boldsymbol{As} + \boldsymbol{e}) + e_2 + m \\
&= \boldsymbol{rAs} + e_3 + e_2 + m \\
\boldsymbol{us} &= (\boldsymbol{rA} + \boldsymbol{e}_1)\boldsymbol{s} \\
&= \boldsymbol{rAs} + e_4
\end{aligned}
$$

무시할 수 있는 작은 noise 값
(메시지를 복구하려면 error를 선택하는 분포가 중요함)

$$\therefore v - \boldsymbol{us} = \boldsymbol{rAs} + e_3 + e_2 + m - (\boldsymbol{rAs} + e_4) = m + (e_3 + e_2 - e_4) \approx m$$

KEM Cryptosystem

Alice

$R = \mathbb{Z}_q[X](X^n + 1), \chi \leftarrow R$

Bob

Key generation

Generate $\boldsymbol{A} \in R^{k \times k}$,

$\boldsymbol{s}, \boldsymbol{e} \leftarrow \chi^k + \boldsymbol{e} = \boldsymbol{t}$

$\boldsymbol{As} + \boldsymbol{e} = \boldsymbol{t}$

$pk : \boldsymbol{A}, \boldsymbol{t}$

$sk : \boldsymbol{s}$ ($\boldsymbol{e}$: secret)

Send $(\boldsymbol{A}, \boldsymbol{t})$

Encapsulation

Generate $m \in \{0,1\}^n$,

$\boldsymbol{r}, \boldsymbol{e}_1 \leftarrow \chi^k, e_2 \leftarrow \chi$

$\boldsymbol{u} = \boldsymbol{rA} + \boldsymbol{e_1}$,

$v = \boldsymbol{rt} + e_2 + m$

$c = (\boldsymbol{u} \parallel v)$

$K' = H(pk \parallel m)$

$K = H(K' \parallel c)$

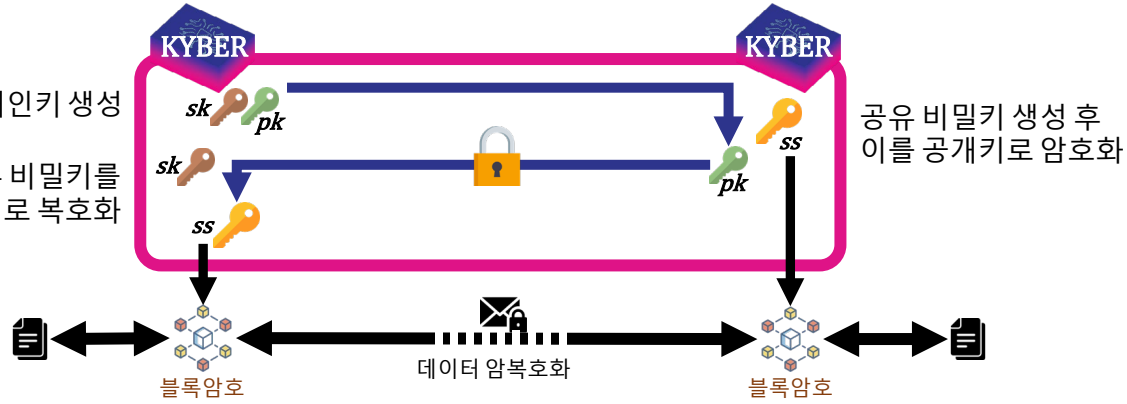
Send c

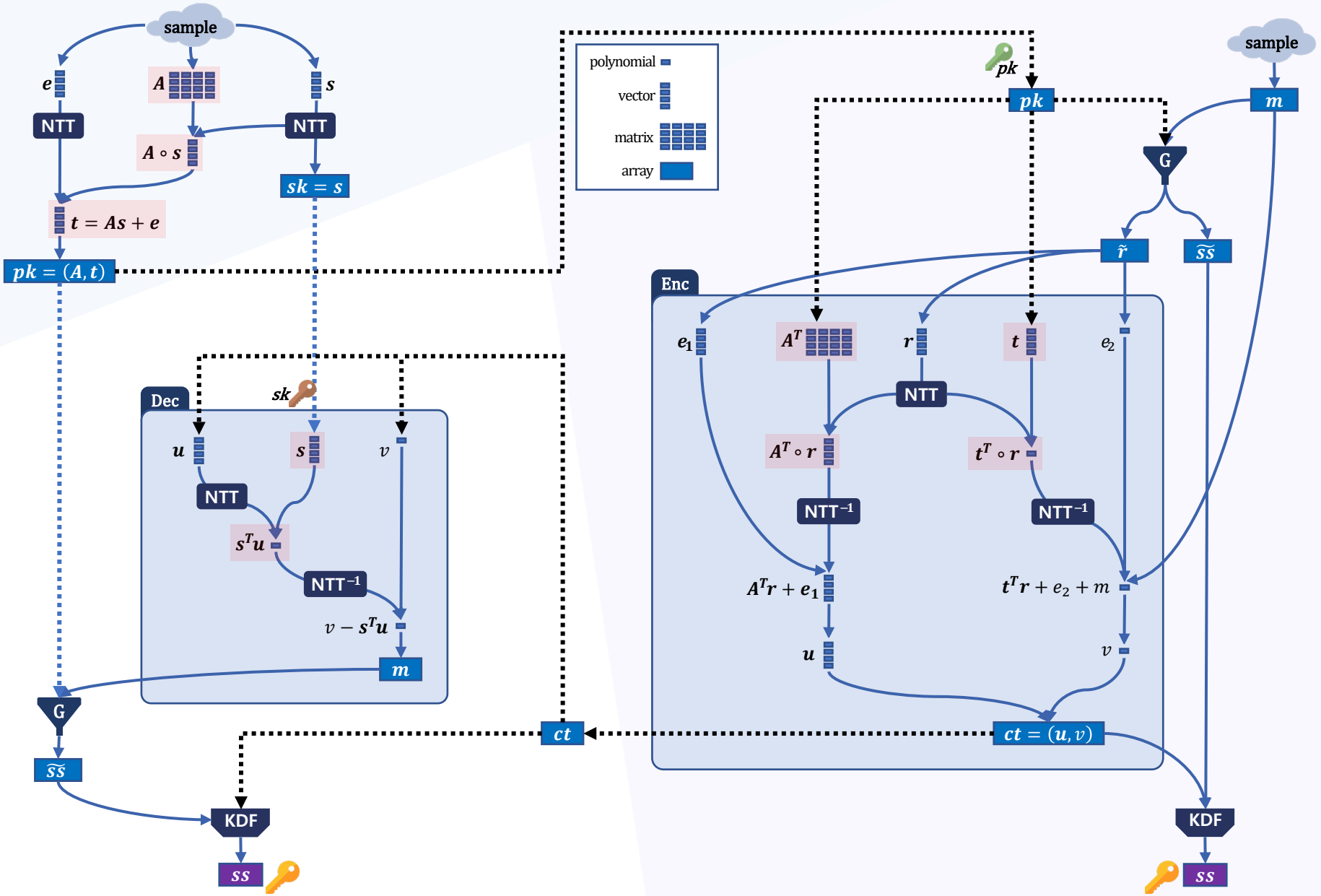
Decapsulation

$v - \boldsymbol{us} = m + e' \approx m$

$K' = H(pk \parallel m)$

$K = H(K' \parallel c)$



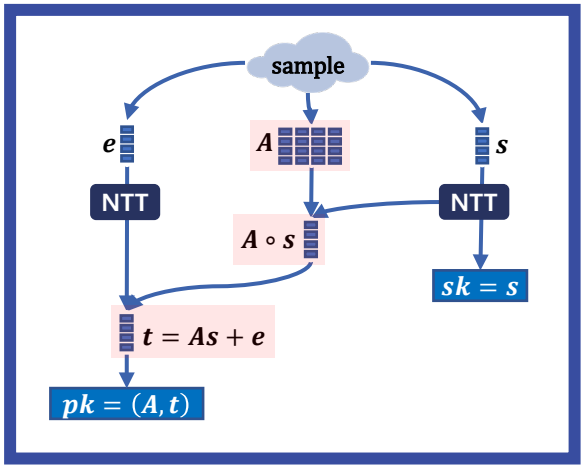

sample
e
A
s
NTT
NTT
A ∘ s
sk = s
t = As + e
pk = (A, t)
polynomial
vector
matrix
array
pk
pk
sample
m
G
r̃
s̃s
Enc
e1
A^T
r
t
e2
NTT
A^T ∘ r
t^T ∘ r
NTT^-1
NTT^-1
A^T r + e1
t^T r + e2 + m
u
v
ct = (u, v)
Dec
sk
u
s
v
NTT
s^T u
NTT^-1
v − s^T u
m
G
s̃s
ct
KDF
KDF
ss
ss

Key generation

Algorithm 6: KYBER.CCAKEM.KeyGen(): key generation

Output: Public key $pk \in \mathcal{B}^{12kn/8+32}$,
Secret key $sk \in \mathcal{B}^{24kn/8+96}$

1: $z \leftarrow \mathcal{B}^{32}$
2: $(pk, sk') := \text{KYBER.CPAPKE.KeyGen}()$
3: $sk := (sk' \parallel pk \parallel \mathbf{H}(pk) \parallel z)$ H : sha3_256
4: **return** (pk, sk)



Algorithm 7: KYBER.CPAPKE.KeyGen(): key generation

Output: Public key $pk \in \mathcal{B}^{12kn/8+32}$,
Secret key $sk \in \mathcal{B}^{12kn/8}$

1: $d \leftarrow \mathcal{B}^{32}$
2: $(\rho, \sigma) := \mathbf{G}(d)$ G : sha3_512
3: $N := 0$
4: **for** $i = 0$ **to** $k-1$ **do**
5: | **for** $j = 0$ **to** $k-1$ **do**
6: | | $\hat{\mathbf{A}}[i][j] := \text{Parse}(\text{XOF}(\rho, j, i))$
7: | **end**
8: **end**
9: **for** $i = 0$ **to** $k-1$ **do**
10: | $\mathbf{s}[i] := \text{CBD}_{\eta_1}(\text{PRF}(\sigma, N))$
11: | $N := N + 1$
12: **end**
13: **for** $i = 0$ **to** $k-1$ **do**
14: | $\mathbf{e}[i] := \text{CBD}_{\eta_1}(\text{PRF}(\sigma, N))$
15: | $N := N + 1$
16: **end**
17: $\hat{\mathbf{s}} := \text{NTT}(\mathbf{s})$
18: $\hat{\mathbf{e}} := \text{NTT}(\mathbf{e})$
19: $\hat{\mathbf{t}} := \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$
20: $pk := (\text{Encode}_{12}(\hat{\mathbf{t}} \mod^+ q) \parallel \rho)$
21: $sk := \text{Encode}_{12}(\hat{\mathbf{s}} \mod^+ q)$
22: **return** (pk, sk)

NTT domain으로부터 matrix $\hat{\boldsymbol{A}} \in R_q^{k \times k}$ 생성
NTT-representation sampling

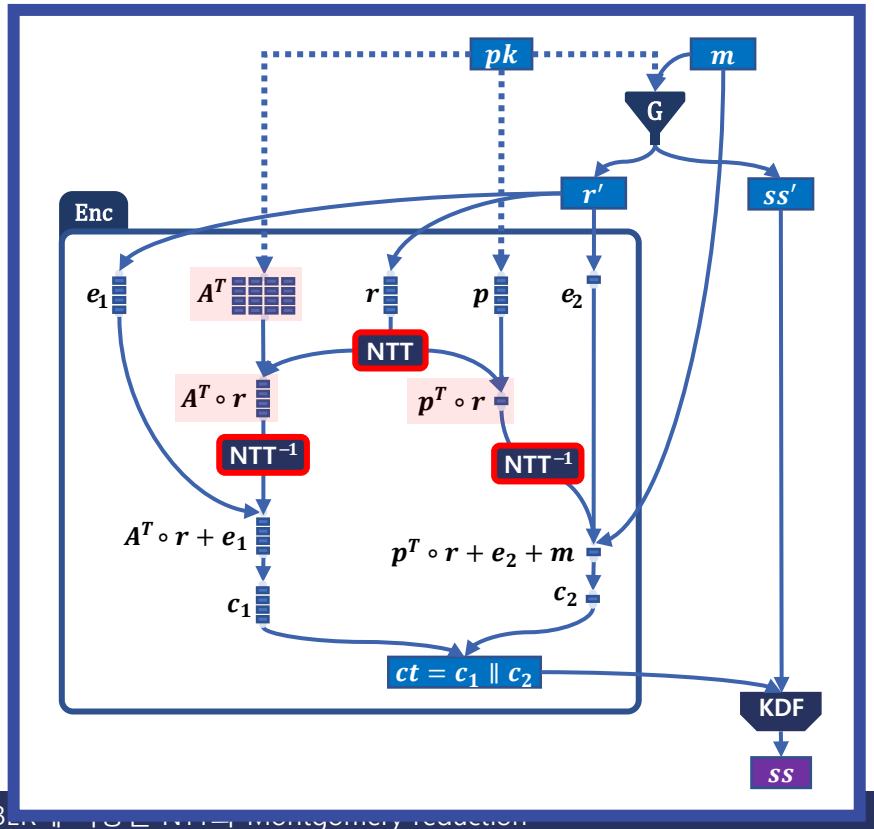
Encapsulation

Algorithm 11: KYBER.CCAKEM.Enc(pk): encapsulation

Input : Public key $pk \in \mathcal{B}^{12kn/8+32}$

Output: Shared key $K \in \mathcal{B}^*$,
Ciphertext $c \in \mathcal{B}^{d_u kn/8+d_v n/8}$

1: $m \leftarrow \mathcal{B}^{32}$
2: $m \leftarrow \mathbf{H}(m)$
3: $(\overline{K}, r) := \mathbf{G}(m \parallel \mathbf{H}(pk))$
4: $c := \text{KYBER.CPAPKE.Enc}(pk, m, r)$
5: $K := \mathsf{KDF}(\overline{K} \parallel \mathbf{H}(c))$
6: **return** (c, K)



Algorithm 12: KYBER.CPAPKE.Enc(pk, m, r): encryption

Input : Public key $pk \in \mathcal{B}^{12kn/8+32}$,
Message $m \in \mathcal{B}^{32}$,
Random coins $r \in \mathcal{B}^{32}$

Output: Ciphertext $c \in \mathcal{B}^{d_u kn/8+d_v n/8}$

1: $N := 0$
2: $\hat{\mathbf{t}} := \mathsf{Decode}_{12}(pk)$
3: $\rho := pk + 12kn/8$
4: **for** $i = 0$ **to** $k-1$ **do**
5: | **for** $j = 0$ **to** $k-1$ **do**
6: | | $\hat{\mathbf{A}}^T[i][j] := \mathsf{Parse}(\mathsf{XOF}(\rho, i, j))$ — $\hat{\boldsymbol{A}} \in R_q^{k \times k}$
7: | **end**
8: **end** — NTT-representation
9: **for** $i = 0$ **to** $k-1$ **do**
10: | $\mathbf{r}[i] := \mathsf{CBD}_{\eta_1}(\mathsf{PRF}(r, N))$ — $\boldsymbol{r} \in R_q^k$
11: | $N := N + 1$
12: **end**
13: **for** $i = 0$ **to** $k-1$ **do**
14: | $\mathbf{e}_1[i] := \mathsf{CBD}_{\eta_2}(\mathsf{PRF}(r, N))$ — $\boldsymbol{e}_1 \in R_q^k$
15: | $N := N + 1$
16: **end**
17: $e_2[i] := \mathsf{CBD}_{\eta_2}(\mathsf{PRF}(r, N))$ — $e_2 \in R_q$
18: $\hat{\mathbf{r}} := \mathsf{NTT}(\mathbf{r})$
19: $\mathbf{u} := \mathsf{NTT}^{-1}(\hat{\mathbf{A}}^T \circ \hat{\mathbf{r}}) + \mathbf{e}_1$
20: $v := \mathsf{NTT}^{-1}(\hat{\mathbf{t}}^T \circ \hat{\mathbf{r}}) + e_2 + \mathsf{Decompress}_q(\mathsf{Decode}_1(m), 1)$
21: $c_1 := \mathsf{Encode}_{d_u}(\mathsf{Compress}_q(\mathbf{u}, d_u))$
22: $c_2 := \mathsf{Encode}_{d_u}(\mathsf{Compress}_q(v, d_v))$
23: **return** $c = (c_1 \parallel c_2)$

Decapsulation

Algorithm 13: KYBER.CCAKEM.Dec(c, sk): decapsulation

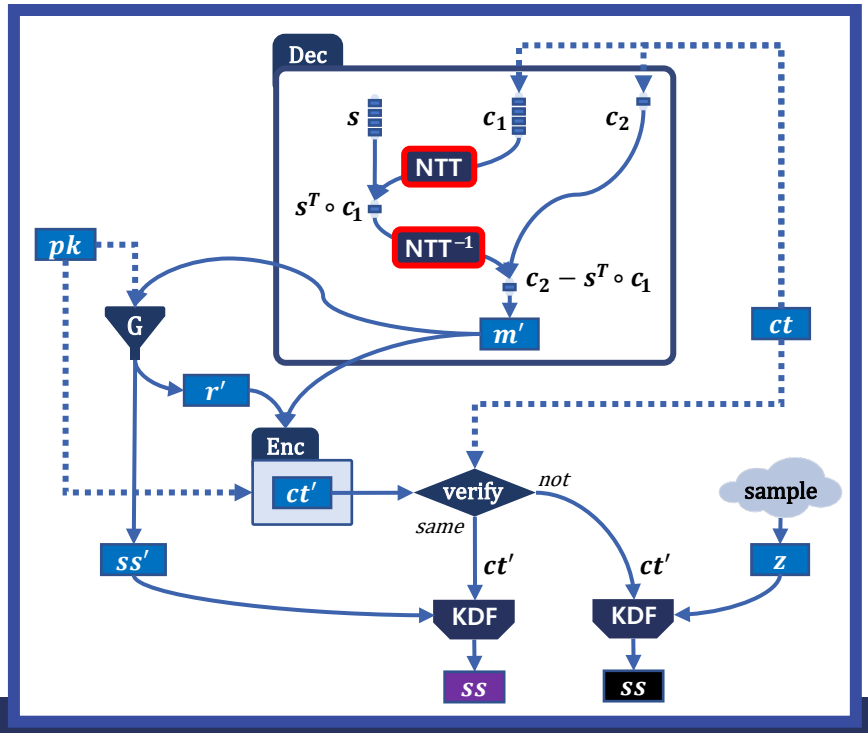
Input : Ciphertext $c \in \mathcal{B}^{d_u kn/8+d_v n/8}$
Output: Secret key $sk \in \mathcal{B}^{24kn/8+96}$,
Shared key $K \in \mathcal{B}^*$

1: $pk := sk + 12 \cdot k \cdot n/8$
2: $h := sk + 24 \cdot k \cdot n/8 + 32 \in \mathcal{B}^{32}$
3: $z := sk + 24 \cdot k \cdot n/8 + 64$
4: $m' := \text{KYBER.CPAPKE.Dec}(\mathbf{s}, (\approx, v))$
5: $(\overline{K}', r') := \mathbf{G}(m' \parallel h)$
6: **if** $c = c'$ **then**
7: **return** $K := \text{KDF}(\overline{K}' \parallel \mathbb{H}(c))$
8: **else**
9: **return** $K := \text{KDF}(z \parallel \mathbb{H}(c))$
10: **end**
11: **return** K

Algorithm 14: KYBER.CPAPKE.Dec(sk, c): decryption

Input : Secret key $sk \in \mathcal{B}^{12kn/8}$,
Ciphertext $c \in \mathcal{B}^{d_u kn/8+d_v n/8}$
Output: Message $m \in \mathcal{B}^{32}$

1: $\mathbf{u} := \text{Decompress}_q(\text{Decode}_{d_u}(c), d_u)$
2: $v := \text{Decompress}_q(\text{Decode}_{d_v}(c + d_u kn/8), d_v)$
3: $\widehat{\mathbf{s}} := \text{Decode}_{12}(sk)$
4: $m :=$ $\text{Encode}_1(\text{Compress}_q(v - \text{NTT}^{-1}(\widehat{\mathbf{s}}^T \circ \text{NTT}(\mathbf{u})), 1))$
5: **return** m



다항식 환 $R_q = \mathbb{Z}_q[x]/\langle x^N + 1\rangle, N = 256, = 2^8, q = 3329$를 사용하는
MLWE(Module Learning With Error) 문제에 기반한 KEM(Key Encapsulation Mechanism)

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

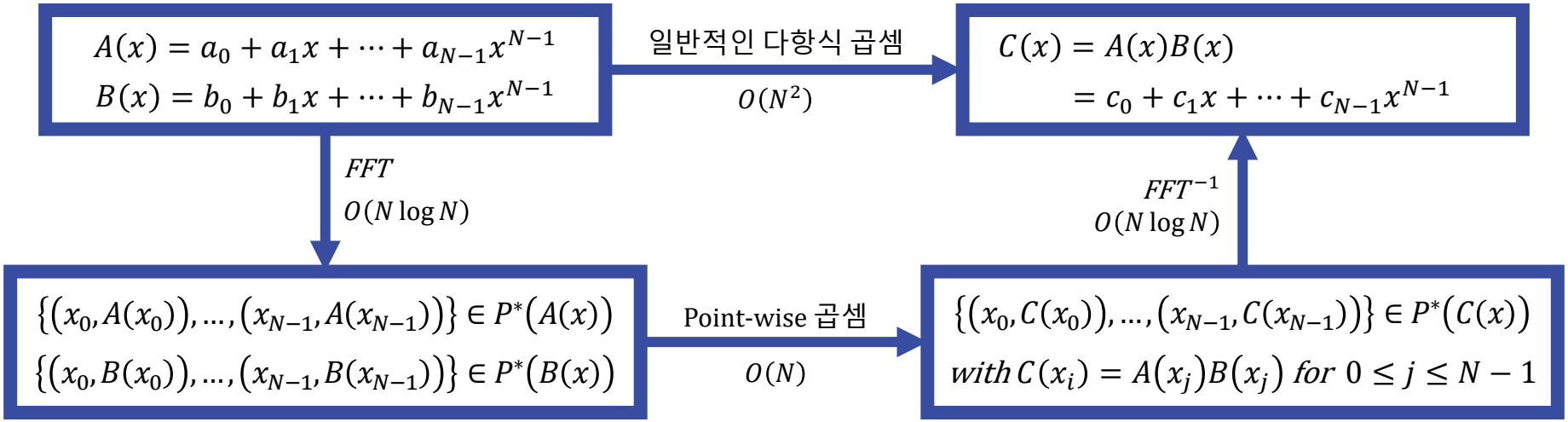
Fast Fourier Transformation(FFT)

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

FFT를 활용한 다항식 곱셈은 서로 다른 N개의 point로 $N-1$차 다항식을 표현할 수 있다는 아이디어에서 시작함 (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
(*다항식 환 $R_q = \mathbb{Z}_q[x]/\langle x^N + 1\rangle$ 위에서의 다항식 곱셈이기 때문*)

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 위에서 FFT를 이용한 다항식 곱셈

Fast Fourier Transformation

알고리즘을 설계하는 과정의 핵심은 어떻게 point를 선택할 것인가로
하나의 x에서 얻은 $A(x), B(x)$의 값을 반복해서 사용할 수 있도록 point를 선택함

가장 직관적인 방식은 우함수와 기함수의 성질을 활용하는 것임

임의의 다항식 $P(x)$에 대하여
$P(x)$가 우함수인 경우 $\big(x, P(x)\big)$를 구하면 $\big(-x, P(x)\big)$가 또다른 point가 되고,
$P(x)$가 기함수인 경우 $\big(x, P(x)\big)$를 구하면 $\big(-x, -P(x)\big)$가 또다른 point가 됨

이는 N차 다항식을 얻기 위하여 N개의 point를 구하는 대신 $N/2$개의 point만 구해도 되도록 만듦

KYBER와 일반적인 응용 환경을 고려하여 이후 나오는 N은 $2^t (t \in \mathbb{Z}^+)$라고 가정

일반적인 다항식 $P(x)$에 이러한 point 선택을 적용하려면 항들을 분리해야 함
$P(x)$의 짝수 차수 항들을 모은 다항식을 $P_e(x)$라고 하고,
홀수 차수 항들을 모은 다항식을 $P_o'(x)$라고 하면

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

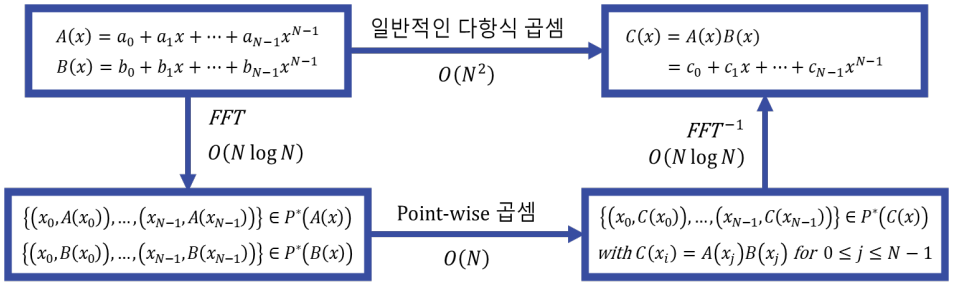
그리고 $P_o'(x)$를 x로 묶으면 $P(x) = P_e(x) + xP_o(x)$를 얻을 수 있음
이때 $P(x)$는 다음 성질을 만족함

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

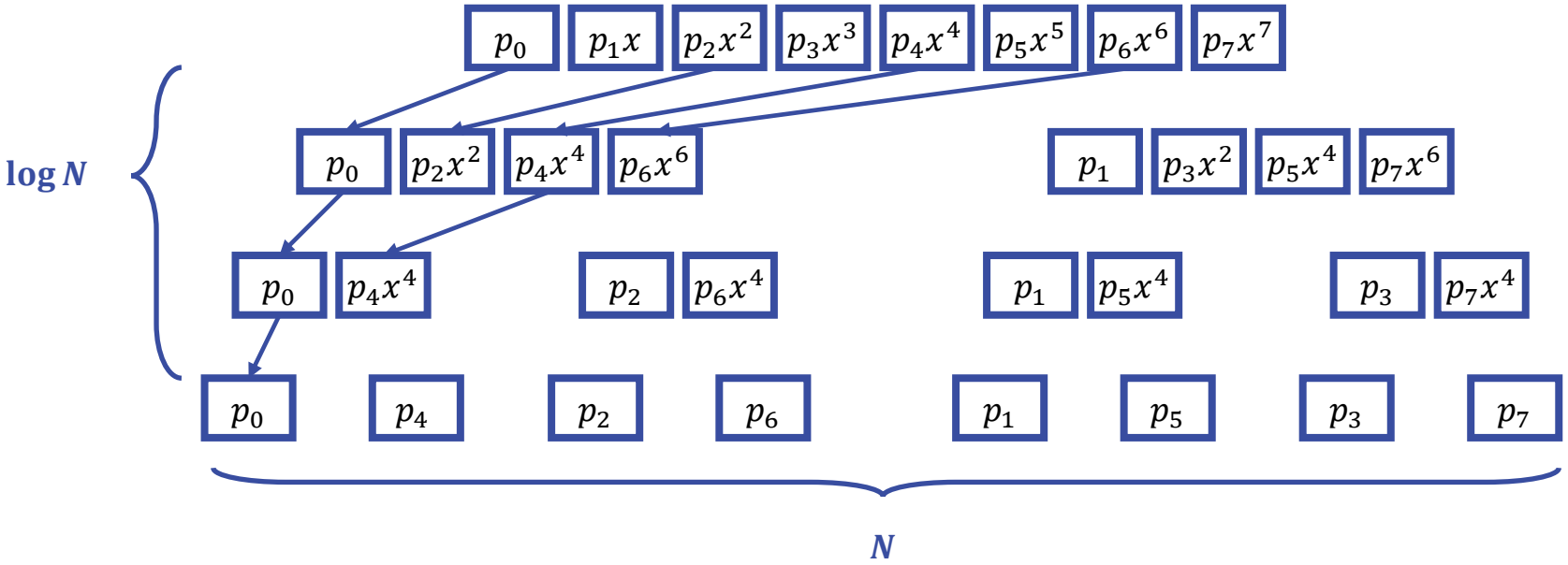
그리고 $P_e(x)$와 $P_o(x)$는 짝수 차수 항들로만 이루어져 있으므로
x^2을 변수로 갖는 다항식으로 생각할 수 있음
이렇게 변수를 치환하면 degree가 $\deg(P_e(x)) = N-1, \deg(P_o(x)) = N-1$에서
$\deg(P_e(x^2)) = \frac{N}{2} - 1, \deg(P_o(x^2)) = \frac{N}{2} - 1$로 줄어듦

$P_e(x^2), P_o(x^2)$을 구하기 위해서는 각각 $N/2$개의 서로 다른 point를 찾으면 되고,
$P(x)$는 $N/2 + N/2$개의 point로 구할 수 있음

이를 재귀적으로 구현하면 하위 과정으로 얻은 $P_e(x), P_o(x)$가 $N = 1$이 될 때까지 수행함으로써 $O(N \log N)$만에 N개의 point를 구할 수 있음 (N개의 root of unity)



$$P(x) = p_0 + p_1x + \cdots + p_7x^7$$
$$= (p_0 + p_2x^2 + p_4x^4 + p_6x^6) + x(p_1 + p_3x^2 + p_5x^4 + p_7x^6) = P_e(x^2) + xP_o(x^2)$$



Fast Fourier Transformation

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

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

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

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

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

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

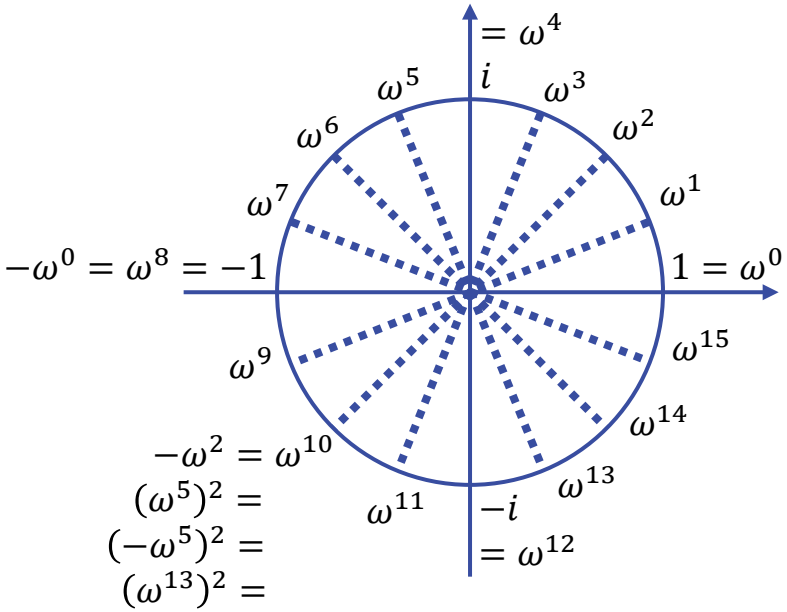
ω^j와 $-\omega^{j+N/2}$는 $\pm x_j$ pair임

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)



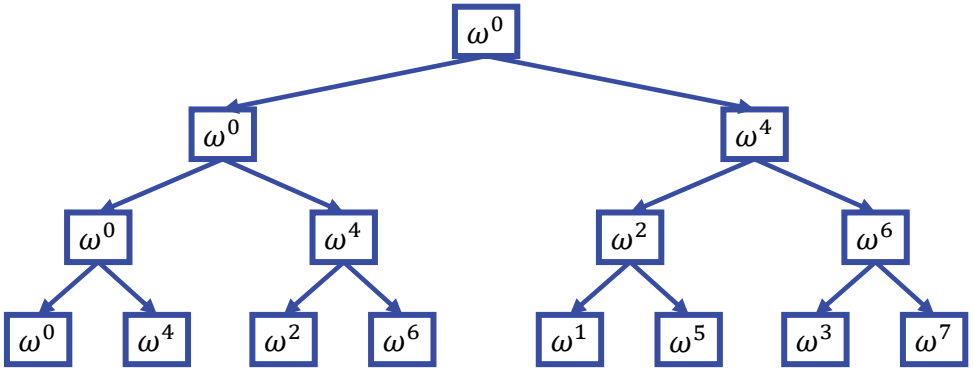
$$P(\omega) = p_0 + p_1\omega + \cdots + p_{N-1}\omega^{N-1}$$
$$= (p_0 + p_2\omega^2 + \cdots + p_{N-2}\omega^{N-2}) + \omega(p_1 + p_3\omega^2 + \cdots + p_{N-1}\omega^{N-2}) = P_e(\omega^2) + \omega P_o(\omega^2)$$

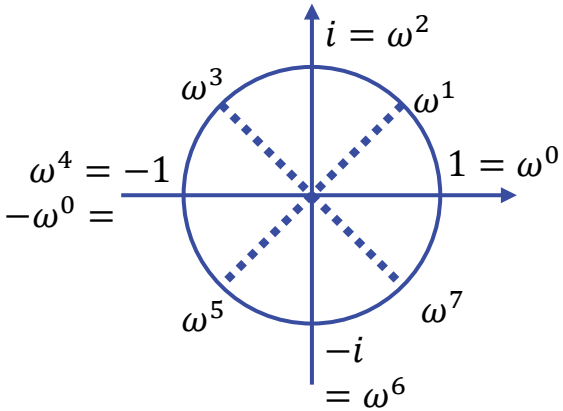
$$P(\omega) = P_e(\omega^2) + \omega P_o(\omega^2)$$

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

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

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





FFT 알고리즘은

$N = 1$이면 $P(1)$을 반환하고

$N > 1$이면 입력 다항식의 절반의 point에 대한 다항식에 FFT 알고리즘을 수행함

FFT는 $O(N^2)$의 연산 복잡도를 갖는 DFT를 $O(N \log N)$의 연산 복잡도로 계산하는 알고리즘

$$DFT\big(A(x)\big) = c_0 + c_1 x + \cdots + c_{N-1}x^{N-1} \text{ where } c_i = \sum_{j=0}^{N-1} a_j \omega^{ij}$$

$$c_i = \sum_{j=0}^{N-1} a_j \omega^{ij} = \sum_{k=0}^{N/2-1} a_{2k}\omega^{2ki} + \sum_{k=0}^{N/2-1} a_{2k+1}\omega^{(2k+1)i}$$

$$= \sum_{k=0}^{N/2-1} a_{2k}\omega^{2ki} + \omega^i \sum_{k=0}^{N/2-1} a_{2k+1}\omega^{2ki}$$

$U_i \coloneqq \sum_{k=0}^{N/2-1} a_{2k}\omega^{2ki}, V_i \coloneqq \sum_{k=0}^{N/2-1} a_{2k+1}\omega^{2ki}$ 라고 하면, $c_i = U_i + \omega^i V_i$

이때 $c_{i+N/2} = U_i - \omega^i V_i$

증명 : 우선 $\omega^{2k(i+N/2)} = \omega^{2ki+kN} = \omega^{2ki}(\omega^N)^k = \omega^{2ki}$.

따라서 $U_{i+N/2} = \sum_{k=0}^{N/2-1} a_{2k}\omega^{2k(i+N/2)} = \sum_{k=0}^{N/2-1} a_{2k}\omega^{2ki} = U_i$

비슷하게 $V_{i+N/2} = \sum_{k=0}^{N/2-1} a_{2k+1}\omega^{2k(i+N/2)} = \sum_{k=0}^{N/2-1} a_{2k+1}\omega^{2ki} = V_i$.

그리고 $\omega^{i+N/2} = \omega^i \omega^{N/2} = -\omega^i$.

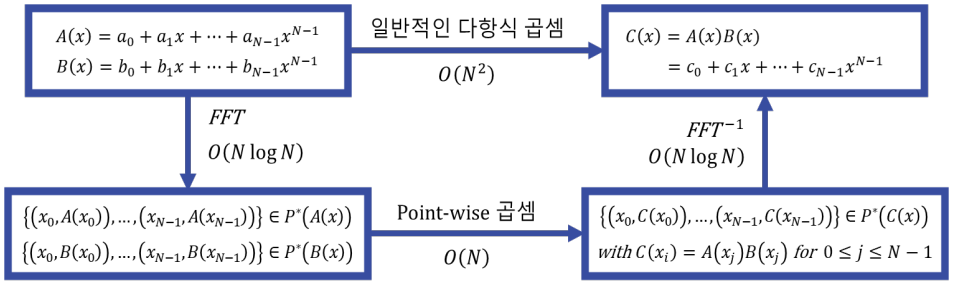
결과적으로 $c_{i+N/2} = U_{i+N/2} + \omega^{i+N/2}V_{i+N/2} = U_i - \omega^i V_i$. ■

Algorithm 1: FFT

Input : Polynomial $P(x) = \sum_{j=0}^{N-1} p_j x^j$

Output: Point value $[y_0, y_1, \ldots, y_{N-1}]$

```
n = len(P(x))                                   ▷ n = N = 2^t
if n = 1 then
    return P(x)                                 ▷ p_0 ∈ Z_q
end
ω ← e^{2πi/n}
P_e = [p_0, p_2, ..., p_{n-2}]
P_o = [p_1, p_3, ..., p_{n-1}]
y_e = FFT(P_e)
y_o = FFT(P_o)
y ← [0]_n
for j = 0 to n/2 − 1 do
    y[j] = y_e[j] + ω^j y_o[j]
    y[j + n/2] = y_e[j] − ω^j y_o[j]
end
return y
```



FFT 역연산 구하기

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

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

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

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

FFT 역연산 구하기

$$P(\boldsymbol{\omega}) = DFT \circ \boldsymbol{p} \text{ for point } \omega^0, \omega^1, \dots, \omega^{N-1}$$

$$\begin{pmatrix} P(\omega^0) \\ P(\omega^1) \\ \vdots \\ P(\omega^{N-1}) \end{pmatrix} = \begin{pmatrix} 1 & \omega^0 & \omega^{0\cdot 1} & \cdots & \omega^{0(N-1)} \\ 1 & \omega^1 & \omega^{1\cdot 1} & \cdots & \omega^{1(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{(N-1)\cdot 1} & \cdots & \omega^{(N-1)(N-1)} \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_{N-1} \end{pmatrix}$$

$\omega^j \rightarrow \frac{1}{N}\omega^{-j} = \frac{1}{N}\omega^{j+N/2}$

$$\boldsymbol{p} = DFT^{-1} \circ P(\boldsymbol{\omega})$$

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

Algorithm 1: FFT

Input : Polynomial $P(x) = \sum_{j=0}^{N-1} p_j x^j$
Output: Point value $[y_0, y_1, \ldots, y_{N-1}]$

1: $n = len(P(x))$ ▷
2: **if** $n = 1$ **then**
3: **return** $P(x)$
4: **end**
5: $\omega \leftarrow e^{2\pi i/n}$
6: $P_e = [p_0, p_2, \ldots, p_{n-2}]$
7: $P_o = [p_1, p_3, \ldots, p_{n-1}]$
8: $y_e = \text{FFT}(P_e)$
9: $y_o = \text{FFT}(P_o)$
10: $y \leftarrow [0]_n$
11: **for** $j = 0$ **to** $n/2 - 1$ **do**
12: $y[j] = y_e[j] + \omega^j y_o[j]$
13: $y[j + n/2] = y_e[j] - \omega^j y_o[j]$
14: **end**
15: **return** y

Algorithm 2: FFT^{-1}

Input : Point value $[y_0, y_1, \ldots, y_{N-1}]$
Output: Polynomial $P(x) = \sum_{j=0}^{N-1} p_j x^j$

1: $n = len(P(x))$ ▷ $n = N = 2^t$
2: **if** $n = 1$ **then**
3: **return** $P(x)$ ▷ $p_0 \in \mathbb{Z}_q$
4: **end**
5: $\omega \leftarrow \frac{1}{n} e^{-2\pi i/n}$
6: $P_e = [p_0, p_2, \ldots, p_{n-2}]$
7: $P_o = [p_1, p_3, \ldots, p_{n-1}]$
8: $y_e = \text{FFT}^{-1}(P_e)$
9: $y_o = \text{FFT}^{-1}(P_o)$
10: $y \leftarrow [0]_n$
11: **for** $j = 0$ **to** $n/2 - 1$ **do**
12: $y[j] = y_e[j] + \omega^j y_o[j]$
13: $y[j + n/2] = y_e[j] - \omega^j y_o[j]$
14: **end**
15: **return** y

DFT(Discrete Fourier Transformation)는

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

임의의 다항식 $A(x) = a_0 + a_1 x + \cdots + a_{N-1}x^{N-1} \in R_q^-$의

DFT $DFT\big(A(x)\big)$와 역DFT $DFT^{-1}\big(A(x)\big)$는 다음과 같이 계산함

$$DFT\big(A(x)\big) = c_0 + c_1 x + \cdots + c_{N-1}x^{N-1} \quad \text{where } c_i = \sum_{j=0}^{N-1} a_j \omega^{ij}$$

$$DFT^{-1}\big(A(x)\big) = d_0 + d_1 x + \cdots + d_{N-1}x^{N-1} \text{ where } d_i = N^{-1}\sum_{j=0}^{N-1} a_j \omega^{-ij}$$

DFT를 활용한 다항식 곱셈은 다항식 계수별 곱셈을 이용해 R_q^-에서 연산을 수행함

연산 복잡도는 $O(N)$

임의의 다항식 $A(x) = a_0 + a_1 x + \cdots + a_{N-1}x^{N-1} \in R_q^-$ 와

$B(x) = b_0 + b_1 x + \cdots + b_{N-1}x^{N-1} \in R_q^-$ 의 곱셈 연산은 다음과 같이 계산함

$$A(x)B(x) = DFT^{-1}\Big(DFT\big(A(x)\big) \circ DFT\big(B(x)\big)\Big)$$

Number Theoretic Transformation

NTT(Number Theoretic Transformation)는
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

R^+의 원소는 $G(x) \coloneqq \sum_{i=0}^{N-1} g^i x^i$와의 계수별 곱셈을 이용해 R_q^-의 원소로 변환함
역으로 R_q^-의 원소는 $G^{-1}(x) \coloneqq \sum_{i=0}^{N-1} g^{-i} x^i$와의 계수별 곱셈을 이용해 R^+의 원소로 변환함

$g = 17$ (first primitive 256-th roots of unity modulo $q = 3329$; $g^{256} = 1 \bmod q$)
$\omega = 289 = g^2 = (17)^2$

임의의 다항식 $A(x) = a_0 + a_1 x + \cdots + a_{N-1} x^{N-1} \in R^+$와
$B(x) = b_0 + b_1 x + \cdots + b_{N-1} x^{N-1} \in R^+$의 곱셈 연산은 다음과 같이 계산함

$$A(x)B(x) = DFT^{-1}\Big(DFT\big((A(x) \circ G(x)\big) \circ DFT\big(B(x) \circ G(x)\big)\Big) \circ G^{-1}(x)$$

$$A(x)B(x) = DFT^{-1}\left(DFT\big((A(x) \circ G(x)\big) \circ DFT\big(B(x) \circ G(x)\big)\right) \circ G^{-1}(x)$$

임의의 다항식 $C(x) = c_0 + c_1x + \cdots + c_{N-1}x^{N-1} \in R^+$,

$D(x) = d_0 + d_1x + \cdots + d_{N-1}x^{N-1} \in R_q^-$에 대하여

NTT와 NTT^{-1}는 다음과 같이 정의함

$$NTT\big(C(x)\big) \coloneqq DFT\big(C(x) \circ G(x)\big)$$

$$NTT^{-1}\big(D(x)\big) \coloneqq DFT^{-1}\big(D(x)\big) \circ G^{-1}(x)$$

즉,

$$A(x)B(x) = NTT^{-1}\left(NTT\big((A(x)\big) \circ NTT\big(B(x)\big)\right)$$

$$NTT\big(C(x)\big) = \sum_{i=0}^{N-1} e_i x^i \quad \text{where} \quad e_i \coloneqq \sum_{j=0}^{N-1} c_j g^j \omega^{ij}$$

$$NTT^{-1}\big(D(x)\big) = \sum_{i=0}^{N-1} f_i x^i \quad \text{where} \quad f_i \coloneqq N^{-1} \sum_{j=0}^{N-1} d_j g^{-j} \omega^{-ij}$$

$$\begin{pmatrix} e_0 \\ e_1 \\ \vdots \\ e_{N-1} \end{pmatrix} = \begin{pmatrix} 1 & \omega^0 & \omega^{0\cdot 1} & \cdots & \omega^{0(N-1)} \\ 1 & \omega^1 & \omega^{1\cdot 1} & \cdots & \omega^{1(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{(N-1)\cdot 1} & \cdots & \omega^{(N-1)(N-1)} \end{pmatrix} \begin{pmatrix} g^0 & 0 & \cdots & 0 \\ 0 & g^1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & g^{N-1} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{N-1} \end{pmatrix}$$

$$\begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{N-1} \end{pmatrix} = \frac{1}{N} \begin{pmatrix} g^{-0} & 0 & \cdots & 0 \\ 0 & g^{-1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & g^{-(N-1)} \end{pmatrix} \begin{pmatrix} 1 & \omega^{-0} & \omega^{-0\cdot 1} & \cdots & \omega^{-0(N-1)} \\ 1 & \omega^{-1} & \omega^{-1\cdot 1} & \cdots & \omega^{-1(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(N-1)} & \omega^{-(N-1)\cdot 1} & \cdots & \omega^{-(N-1)(N-1)} \end{pmatrix} \begin{pmatrix} d_0 \\ d_1 \\ \vdots \\ d_{N-1} \end{pmatrix}$$

Number Theoretic Transformation

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

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

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

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

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

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

$y_e = p_0 \qquad y_o = p_4$

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

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

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

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

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

$y_e = p_2 \qquad y_o = p_6$

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

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

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

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

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

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

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

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

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

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

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

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

$y_e = p_1 \qquad y_o = p_5$

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

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

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

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

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

$y_e = p_3 \qquad y_o = p_7$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

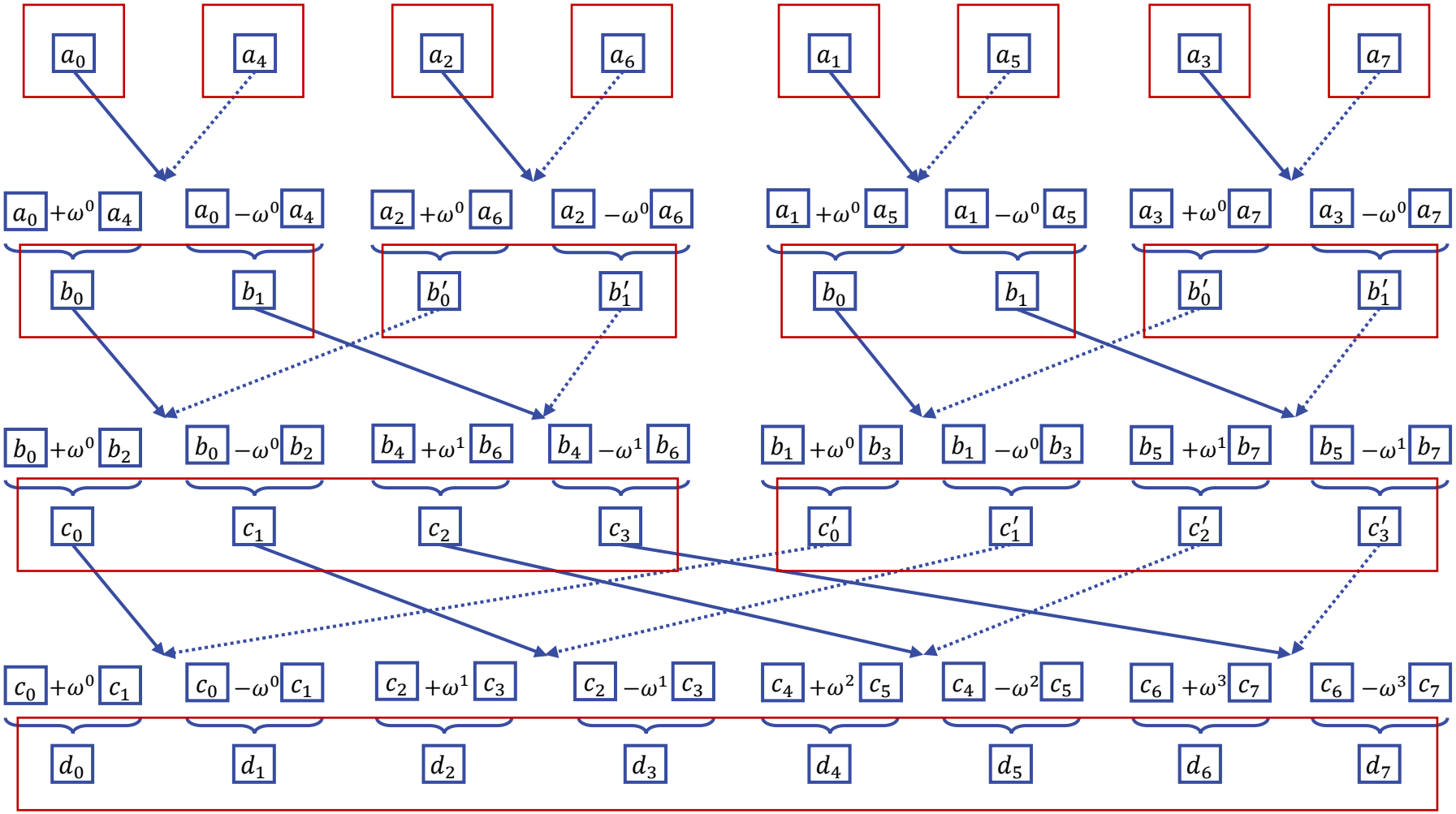
$\boldsymbol{c_7} = y[7] = y_e[3] - w^3 y_o[3] = (\boldsymbol{p_0 - \omega^0 p_4}) - \boldsymbol{\omega^2}(\boldsymbol{p_2 - \omega^0 p_6}) - \boldsymbol{\omega^3}\left((\boldsymbol{p_1 - \omega^0 p_5}) - \boldsymbol{\omega^2}(\boldsymbol{p_3 - \omega^0 p_7})\right)$

Number Theoretic Transformation

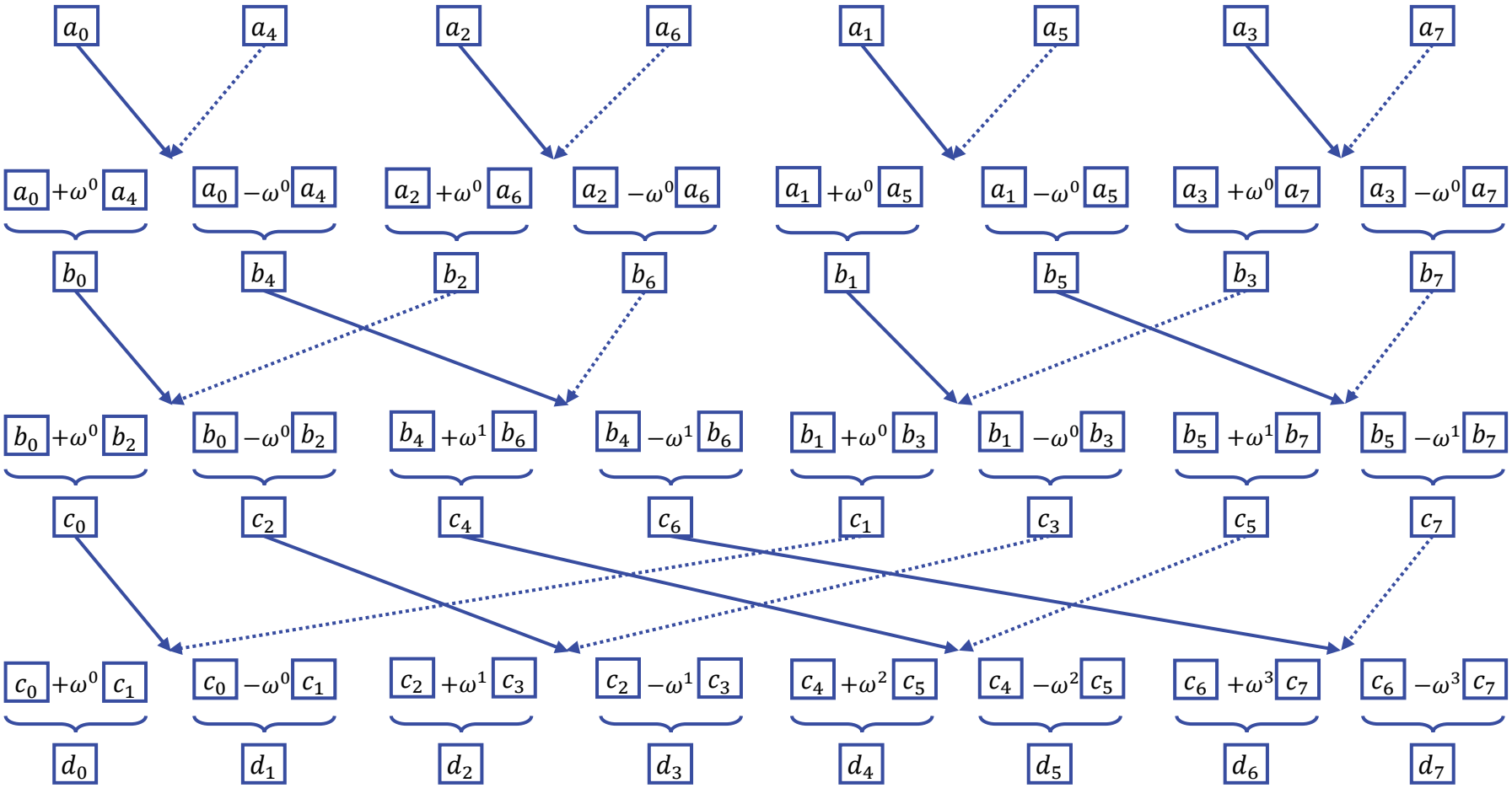
$n = 8, w \leftarrow e^{\frac{2\pi i}{8}} = \omega$

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

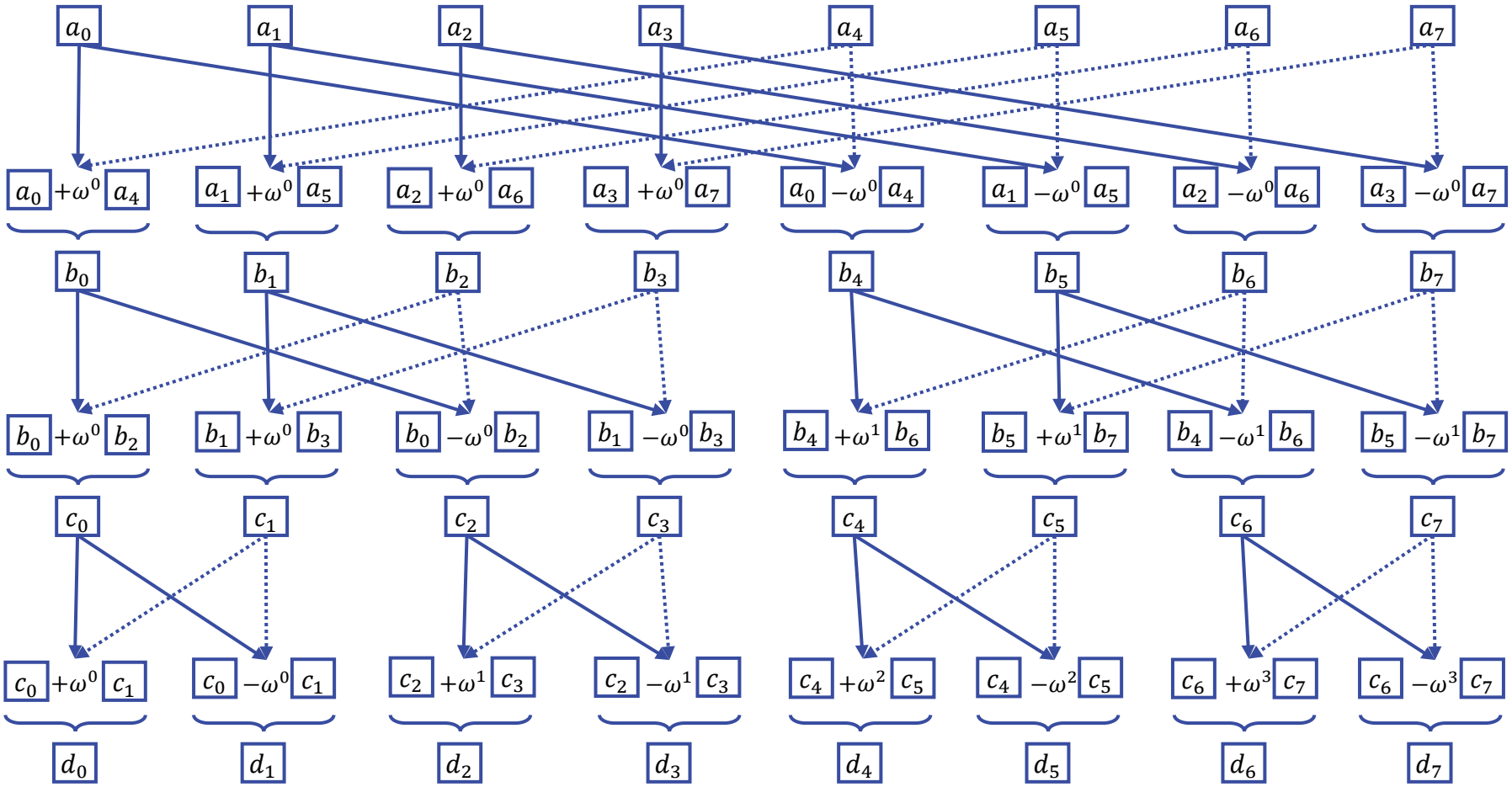
$$
\begin{aligned}
\boldsymbol{c_0} &= (p_0 + \omega^0 p_4) + \omega^0(p_2 + \omega^0 p_6) + \omega^0\big((p_1 + \omega^0 p_5) + \omega^0(p_3 + \omega^0 p_7)\big) \\
&= p_0 + p_4\omega^0 + p_2\omega^0 + p_6(\omega^0)^2 + p_1\omega^0 + p_5(\omega^0)^2 + p_3(\omega^0)^2 + p_7(\omega^0)^3 \\
&= p_0 + p_1\omega^0 + p_2\omega^0 + p_3\omega^0 + p_4\omega^0 + p_5\omega^0 + p_6\omega^0 + p_7\omega^0 = \boldsymbol{P(\omega^0)} \\
\boldsymbol{c_4} &= (p_0 + \omega^0 p_4) + \omega^0(p_2 + \omega^0 p_6) - \omega^0\big((p_1 + \omega^0 p_5) + \omega^0(p_3 + \omega^0 p_7)\big) \\
&= p_0 + p_4\omega^0 + p_2\omega^0 + p_6(\omega^0)^2 - p_1\omega^0 - p_5(\omega^0)^2 - p_3(\omega^0)^2 - p_7(\omega^0)^3 \\
&= p_0 - p_1\omega^0 + p_2\omega^0 - p_3\omega^0 + p_4\omega^0 - p_5\omega^0 + p_6\omega^0 - p_7\omega^0 = \boldsymbol{P(\omega^4)} \\
\boldsymbol{c_1} &= (p_0 - \omega^0 p_4) + \omega^2(p_2 - \omega^0 p_6) + \omega^1\big((p_1 - \omega^0 p_5) + \omega^2(p_3 - \omega^0 p_7)\big) \\
&= p_0 - p_4\omega^0 + p_2\omega^2 - p_6\omega^2 + p_1\omega^1 - p_5\omega^1 + p_3\omega^3 - p_7\omega^3 \\
&= p_0 + p_1\omega^1 + p_2\omega^2 + p_3\omega^3 - p_4\omega^0 - p_5\omega^1 - p_6\omega^2 - p_7\omega^3 = \boldsymbol{P(\omega^1)} \\
\boldsymbol{c_5} &= (p_0 - \omega^0 p_4) + \omega^2(p_2 - \omega^0 p_6) - \omega^1\big((p_1 - \omega^0 p_5) + \omega^2(p_3 - \omega^0 p_7)\big) \\
&= p_0 - p_4\omega^0 + p_2\omega^2 - p_6\omega^2 - p_1\omega^1 + p_5\omega^1 - p_3\omega^3 - p_7\omega^3 \\
&= p_0 - p_1\omega^1 + p_2\omega^2 - p_3\omega^3 - p_4\omega^0 + p_5\omega^1 - p_6\omega^2 - p_7\omega^3 = \boldsymbol{P(\omega^5)} \\
\boldsymbol{c_2} &= (p_0 + \omega^0 p_4) - \omega^0(p_2 + \omega^0 p_6) + \omega^2\big((p_1 + \omega^0 p_5) - \omega^0(p_3 + \omega^0 p_7)\big) \\
&= p_0 + p_4\omega^0 - p_2\omega^0 - p_6\omega^0 + p_1\omega^2 + p_5\omega^2 - p_3\omega^2 - p_7\omega^2 \\
&= p_0 + p_1\omega^2 - p_2\omega^0 - p_3\omega^2 + p_4\omega^0 + p_5\omega^2 - p_6\omega^0 - p_7\omega^2 = \boldsymbol{P(\omega^2)} \\
\boldsymbol{c_6} &= (p_0 + \omega^0 p_4) - \omega^0(p_2 + \omega^0 p_6) - \omega^2\big((p_1 + \omega^0 p_5) - \omega^0(p_3 + \omega^0 p_7)\big) \\
&= p_0 + p_4\omega^0 - p_2\omega^0 - p_6\omega^0 - p_1\omega^2 - p_5\omega^2 + p_3\omega^2 + p_7\omega^2 \\
&= p_0 - p_1\omega^2 - p_2\omega^0 + p_3\omega^2 + p_4\omega^0 - p_5\omega^2 - p_6\omega^0 + p_7\omega^2 = \boldsymbol{P(\omega^6)} \\
\boldsymbol{c_3} &= (p_0 - \omega^0 p_4) - \omega^2(p_2 - \omega^0 p_6) + \omega^3\big((p_1 - \omega^0 p_5) - \omega^2(p_3 - \omega^0 p_7)\big) \\
&= p_0 - p_4\omega^0 - p_2\omega^2 + p_6\omega^2 + p_1\omega^3 - p_5\omega^3 - p_3\omega^5 + p_7\omega^5 \\
&= p_0 + p_1\omega^3 - p_2\omega^2 - p_3\omega^5 - p_4\omega^0 - p_5\omega^3 + p_6\omega^2 + p_7\omega^5 = \boldsymbol{P(\omega^3)} \\
\boldsymbol{c_7} &= (p_0 - \omega^0 p_4) - \omega^2(p_2 - \omega^0 p_6) - \omega^3\big((p_1 - \omega^0 p_5) - \omega^2(p_3 - \omega^0 p_7)\big) \\
&= p_0 - p_4\omega^0 - p_2\omega^2 + p_6\omega^2 - p_1\omega^3 + p_5\omega^3 + p_3\omega^5 - p_7\omega^5 \\
&= p_0 - p_1\omega^3 - p_2\omega^2 + p_3\omega^5 - p_4\omega^0 + p_5\omega^3 + p_6\omega^2 - p_7\omega^5 = \boldsymbol{P(\omega^7)}
\end{aligned}
$$

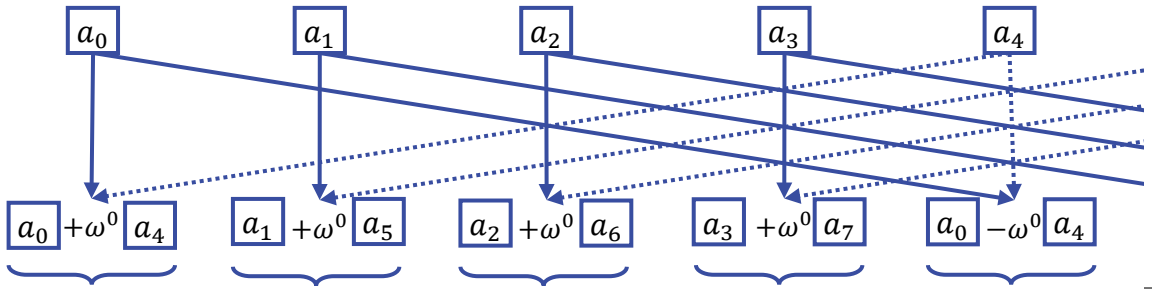

a_0 a_4 a_2 a_6 a_1 a_5 a_3 a_7
$a_0 + \omega^0 a_4$ $a_0 - \omega^0 a_4$ $a_2 + \omega^0 a_6$ $a_2 - \omega^0 a_6$ $a_1 + \omega^0 a_5$ $a_1 - \omega^0 a_5$ $a_3 + \omega^0 a_7$ $a_3 - \omega^0 a_7$
b_0 b_1 b_0' b_1' b_0 b_1 b_0' b_1'
$b_0 + \omega^0 b_2$ $b_0 - \omega^0 b_2$ $b_4 + \omega^1 b_6$ $b_4 - \omega^1 b_6$ $b_1 + \omega^0 b_3$ $b_1 - \omega^0 b_3$ $b_5 + \omega^1 b_7$ $b_5 - \omega^1 b_7$
c_0 c_1 c_2 c_3 c_0' c_1' c_2' c_3'
$c_0 + \omega^0 c_1$ $c_0 - \omega^0 c_1$ $c_2 + \omega^1 c_3$ $c_2 - \omega^1 c_3$ $c_4 + \omega^2 c_5$ $c_4 - \omega^2 c_5$ $c_6 + \omega^3 c_7$ $c_6 - \omega^3 c_7$
d_0 d_1 d_2 d_3 d_4 d_5 d_6 d_7

Number Theoretic Transformation

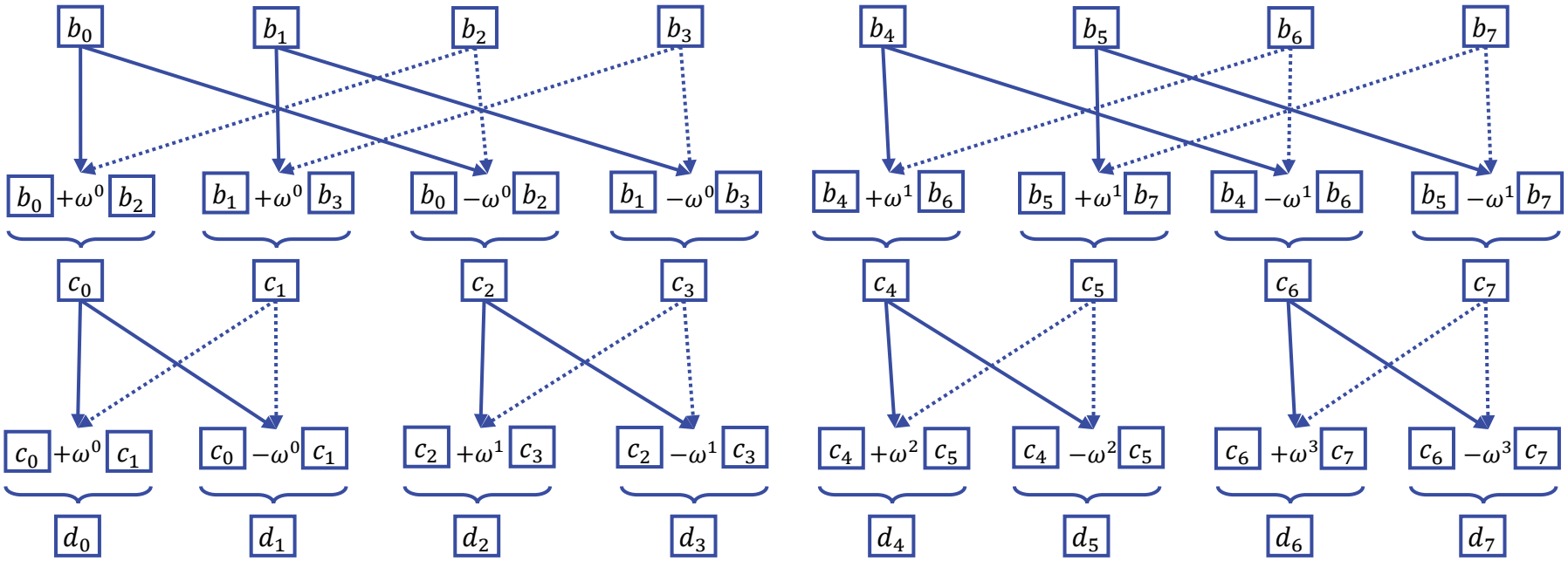


Number Theoretic Transformation






```
k ← 1
for l ← n/2; l > 0; l ← l/2 do
    for s ← 0; s < n; s ← j + l do
        for j ← s; j < s + l; j ← j + 1 do
            t ← ζ_k · c_{j+l}
            c_{j+l} ← c_j − t
            c_j ← c_j + t
        end for
        k ← k + 1
    end for
end for
```



Number Theoretic Transformation

$A(x) = a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$

$A[8] = [a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7]$

l=4

s=0, k=1

j=0 $a_4 = a_0 - \zeta_1 a_4$, $a_0 = a_0 + \zeta_1 a_4$

j=1 $a_5 = a_1 - \zeta_1 a_5$, $a_1 = a_1 + \zeta_1 a_5$

j=2 $a_6 = a_2 - \zeta_1 a_6$, $a_2 = a_2 + \zeta_1 a_6$

j=3 $a_7 = a_3 - \zeta_1 a_7$, $a_3 = a_3 + \zeta_1 a_7$

$\zeta_1 = \omega^0$

$\zeta_2 = \omega^0$

$\zeta_3 = \omega^1$

l=2

s=0 , k=2

j=0 $a_2 = (a_0 + \zeta_1 a_4) - \zeta_2(a_2 + \zeta_1 a_6)$, $a_0 = (a_0 + \zeta_1 a_4) + \zeta_2(a_2 + \zeta_1 a_6)$

j=1 $a_3 = (a_1 + \zeta_1 a_5) - \zeta_2(a_3 + \zeta_1 a_7)$, $a_1 = (a_1 + \zeta_1 a_5) + \zeta_2(a_3 + \zeta_1 a_7)$

s=4, k=3

j=4 $a_6 = (a_0 - \zeta_1 a_4) - \zeta_3(a_2 - \zeta_1 a_6)$, $a_4 = (a_0 - \zeta_1 a_4) + \zeta_3(a_2 - \zeta_1 a_6)$

j=5 $a_7 = (a_1 - \zeta_1 a_5) - \zeta_3(a_3 - \zeta_1 a_7)$, $a_5 = (a_1 - \zeta_1 a_5) + \zeta_3(a_3 - \zeta_1 a_7)$

Algorithm 1 Forward NTT of a polynomial $f = c_0 + c_1X \ldots c_{n-1}X^{n-1} \in \mathbb{Z}_q[X]/(X^n + 1)$ 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

Number Theoretic Transformation

$A(x) = a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$

$A[8] = [a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7]$

l=2

s=0 , k=2

j=0 $\quad a_2 = (a_0 + \zeta_1 a_4) - \zeta_2(a_2 + \zeta_1 a_6)$
$\quad\quad\quad a_0 = (a_0 + \zeta_1 a_4) + \zeta_2(a_2 + \zeta_1 a_6)$

j=1 $\quad a_3 = (a_1 + \zeta_1 a_5) - \zeta_2(a_3 + \zeta_1 a_7)$
$\quad\quad\quad a_1 = (a_1 + \zeta_1 a_5) + \zeta_2(a_3 + \zeta_1 a_7)$

s=4 , k=3

j=4 $\quad a_6 = (a_0 - \zeta_1 a_4) - \zeta_3(a_2 - \zeta_1 a_6)$
$\quad\quad\quad a_4 = (a_0 - \zeta_1 a_4) + \zeta_3(a_2 - \zeta_1 a_6)$

j=5 $\quad a_7 = (a_1 - \zeta_1 a_5) - \zeta_3(a_3 - \zeta_1 a_7)$
$\quad\quad\quad a_5 = (a_1 - \zeta_1 a_5) + \zeta_3(a_3 - \zeta_1 a_7)$

l=1

s=0 , k=4

j=0 $\quad a_1 = \big((a_0 + \zeta_1 a_4) + \zeta_2(a_2 + \zeta_1 a_6)\big) - \zeta_4\big((a_1 + \zeta_1 a_5) + \zeta_2(a_3 + \zeta_1 a_7)\big) \rightarrow y[4]$
$\quad\quad\quad a_0 = \big((a_0 + \zeta_1 a_4) + \zeta_2(a_2 + \zeta_1 a_6)\big) + \zeta_4\big((a_1 + \zeta_1 a_5) + \zeta_2(a_3 + \zeta_1 a_7)\big) \rightarrow y[0]$

s=2 , k=5

j=2 $\quad a_3 = \big((a_0 + \zeta_1 a_4) - \zeta_2(a_2 + \zeta_1 a_6)\big) - \zeta_5\big((a_1 + \zeta_1 a_5) - \zeta_2(a_3 + \zeta_1 a_7)\big) \rightarrow y[5]$
$\quad\quad\quad a_2 = \big((a_0 + \zeta_1 a_4) - \zeta_2(a_2 + \zeta_1 a_6)\big) + \zeta_5\big((a_1 + \zeta_1 a_5) - \zeta_2(a_3 + \zeta_1 a_7)\big) \rightarrow y[1]$

s=4 , k=6

j=4 $\quad a_5 = \big((a_0 - \zeta_1 a_4) + \zeta_3(a_2 - \zeta_1 a_6)\big) - \zeta_6\big((a_1 - \zeta_1 a_5) + \zeta_3(a_3 - \zeta_1 a_7)\big) \rightarrow y[6]$
$\quad\quad\quad a_4 = \big((a_0 - \zeta_1 a_4) + \zeta_3(a_2 - \zeta_1 a_6)\big) + \zeta_6\big((a_1 - \zeta_1 a_5) + \zeta_3(a_3 - \zeta_1 a_7)\big) \rightarrow y[2]$

s=6 , k=7

j=6 $\quad a_7 = \big((a_0 - \zeta_1 a_4) - \zeta_3(a_2 - \zeta_1 a_6)\big) - \zeta_7\big((a_1 - \zeta_1 a_5) - \zeta_3(a_3 - \zeta_1 a_7)\big) \rightarrow y[7]$
$\quad\quad\quad a_6 = \big((a_0 - \zeta_1 a_4) - \zeta_3(a_2 - \zeta_1 a_6)\big) + \zeta_7\big((a_1 - \zeta_1 a_5) - \zeta_3(a_3 - \zeta_1 a_7)\big) \rightarrow y[3]$

$\zeta_1 = \omega^0$

$\zeta_2 = \omega^0$

$\zeta_3 = \omega^1$

$\zeta_4 = \omega^0$

$\zeta_5 = \omega^1$

$\zeta_6 = \omega^2$

$\zeta_7 = \omega^3$

```
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;
      }
    }
  }
}
```

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

  k = 0;
  for(len = 2; len <= 128; len <<= 1) {
    for(start = 0; start < 256; start = j + len) {
      zeta = zetas_inv[k++];
      for(j = start; j < start + len; ++j) {
        t = r[j];
        r[j] = barrett_reduction(t + r[j + len]);
        r[j + len] = t - r[j + len];
        r[j + len] = fqmul(zeta, r[j + len]);
      }
    }
  }

  for(j = 0; j < 256; ++j)
    r[j] = fqmul(r[j], zetas_inv[127]);
}
```

ntt는 외부에서 수행

$$s[j+len] = r[j] - \zeta_k \cdot r[j+len]$$
$$s[j] = r[j] + \zeta_k \cdot r[j+len]$$

$$s'[j] = (s[j] + s[j+len])$$
$$s'[j+len] = \zeta_k^{-1}(s[j] - s[j+len])$$

$$s'[j] \leftarrow N^{-1} \cdot s'[j]$$

```c
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;
      }
    }
  }
}
```

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

  k = 0;
  for(len = 2; len <= 128; len <<= 1) {
    for(start = 0; start < 256; start = j + len) {
      zeta = zetas_inv[k++];
      for(j = start; j < start + len; ++j) {
        t = r[j];
        r[j] = barrett_reduction(t + r[j + len]);
        r[j + len] = t - r[j + len];
        r[j + len] = fqmul(zeta, r[j + len]);
      }
    }
  }

  for(j = 0; j < 256; ++j)
    r[j] = fqmul(r[j], zetas_inv[127]);
}
```

ntt는 외부에서 수행

$$s[j+len] = r[j] - \zeta_k \cdot r[j+len]$$
$$s[j] = r[j] + \zeta_k \cdot r[j+len]$$

$$\begin{aligned} s'[j] &= (s[j] + s[j+len]) \\ &= (r[j] + \zeta_k \cdot r[j+len] + r[j] - \zeta_k \cdot r[j+len]) \\ &= 2r[j] \\ s'[j+len] &= \zeta_k^{-1}(s[j] - s[j+len]) \\ &= \zeta_k^{-1}(r[j] + \zeta_k \cdot r[j+len] - r[j] + \zeta_k \cdot r[j+len]) \\ &= 2r[j+len] \end{aligned}$$

$$s'[j] \leftarrow 2^{-1} \cdot s'[j] = r[j]$$

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: $r' \leftarrow a_1 - t_1$

β: 워드 크기
(프로그램 자료형)

(Line 1)

$a = mq + r'\beta$라고 하면, $\quad a \equiv r'\beta \mod q$

$a\beta^{-1} \equiv r' \mod q$ (r' : *signed Montgomery reduction의 output*)

이 a에 대하여 $\quad a \equiv mq \mod \beta$

$aq^{-1} \equiv m \mod \beta$

그리고 input a의 조건에 의해 $\quad a \equiv a_0 \mod \beta$

따라서 $\quad a_0 q^{-1} \equiv m \mod \beta.$

Montgomery reduction

Algorithm 3 Signed Montgomery reduction

Require: $0 < q < \frac{\beta}{2}$ odd, $-\frac{\beta}{2}q \le a = a_1\beta + a_0 < \frac{\beta}{2}q$ where $0 \le a_0 < \beta$
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: $r' \leftarrow a_1 - t_1$

(Line 2)

$mq = t = t_1\beta + t_0$라고 하면,

$$a - t = a_1\beta + a_0 - (t_1\beta + t_0)$$
$$= (a_1 - t_1)\beta + (a_0 - t_0)$$

그리고 가정들에 의하여

$$a - t = mq + r'\beta - mq$$
$$= r'\beta$$

따라서

$$(a_1 - t_1)\beta + (a_0 - t_0) = r'\beta$$
$$a_0 - t_0 \equiv 0 \mod \beta$$

이때 $a_0, t_0 \in [0, \beta)$이기 때문에

$$a_0 = t_0$$

(Line 3)

즉, $(a_1 - t_1)\beta = r'\beta$ $\Rightarrow$ $a_1 - t_1 = r'$ ■

Montgomery reduction

$g = 17$ (first primitive 256-th roots of unity modulo $q = 3329$; $g^{256} = 1 \bmod q$)

```
void init_ntt() {
  unsigned int i, j, k;
  int16_t tmp[128];

  tmp[0] = MONT;
  for(i = 1; i < 128; ++i)
    tmp[i] = fqmul(tmp[i-1],
                   KYBER_ROOT_OF_UNITY*MONT % KYBER_Q);

  for(i = 0; i < 128; ++i)
    zetas[i] = tmp[tree[i]];
```

tree[4]=[0, 2, 1, 3]

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

For input a, return $R^{-1}a \bmod Q$

Ex: $g^8 = 1 \bmod Q$, $g^2 = \omega$

$\text{tmp}[8/2] = [R, 0, 0, 0]$

$\text{tmp}[8/2] = [R, R*\{(g*R) \bmod Q\}, 0, 0]$

$= [R, gR \bmod Q, 0, 0, 0]$

$\text{tmp}[8/2] = [R, gR \bmod Q, (gR \bmod Q)\{(g*R) \bmod Q\}, 0]$

$= [R, gR \bmod Q, g^2R \bmod Q, 0]$

$\text{tmp}[8/2] = [R, gR \bmod Q, g^2R \bmod Q, (g^2R \bmod Q)\{(g*R) \bmod Q\}]$

$= [R, gR \bmod Q, g^2R \bmod Q, g^3R \bmod Q]$

$\text{zetas}[4] = [R, g^2R \bmod Q, gR \bmod Q, g^3R \bmod Q]$

$= [R, (g\omega^0)^2R \bmod Q, (g\omega^0)^1R \bmod Q, (g\omega^1)^1R \bmod Q]$

$NTT(C(x)) = \sum_{i=0}^{N-1} e_i x^i$ where $e_i := \sum_{j=0}^{N-1} c_j (g\omega^i)^j$

```
#define MONT 2285 // 2^16 mod q
#define KYBER_ROOT_OF_UNITY 17
#define KYBER_Q 3329
```
$=2^8 13+1$

Montgomery reduction

$g = 17$ (first primitive 256-th roots of unity modulo $q = 3329$; $g^{256} = 1 \bmod q$)

```
void init_ntt() {
  unsigned int i, j, k;
  int16_t tmp[128];

  tmp[0] = MONT;
  for(i = 1; i < 128; ++i)
    tmp[i] = fqmul(tmp[i-1],
                   KYBER_ROOT_OF_UNITY*MONT % KYBER_Q);

  for(i = 0; i < 128; ++i)
    zetas[i] = tmp[tree[i]];
```

tree[8]=[0, 4, 2, 6, 1, 5, 3, 7]

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

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;
}
```

For input a, return $R^{-1}a \bmod Q$

Ex: $g^{16} = 1 \bmod Q$, $g^2 = \omega$

$\text{tmp}[16/2] = [R, 0, 0, 0, 0, 0, 0, 0]$

$\text{tmp}[16/2] = [R, R*\{(g*R) \bmod Q\}, 0, 0, 0, 0, 0, 0]$

$= [R, gR \bmod Q, 0, 0, 0, 0, 0, 0, 0]$

$\text{tmp}[16/2] = [R, gR \bmod Q, (gR \bmod Q)\{(g*R) \bmod Q\}, 0, 0, 0, 0, 0]$

$= [R, gR \bmod Q, g^2R \bmod Q, 0, 0, 0, 0, 0]$

$\text{tmp}[16/2] = [R, gR \bmod Q, g^2R \bmod Q, g^3R \bmod Q, \dots, g^7R \bmod Q]$

$\text{zetas}[8] = [R, g^4R \bmod Q, g^2R \bmod Q, g^6R \bmod Q,$

$gR \bmod Q, g^5R \bmod Q, g^3R \bmod Q, g^7R \bmod Q]$

$= [R, (g\omega^0)^4R \bmod Q, (g\omega^0)^2R \bmod Q, (g\omega^1)^2R \bmod Q,$

$(g\omega^0)^1R \bmod Q, (g\omega^2)^1R \bmod Q, (g\omega^1)^1R \bmod Q, (g\omega^3)^1R \bmod Q]$

$NTT(C(x)) = \sum_{i=0}^{N-1} e_i x^i$ where $e_i := \sum_{j=0}^{N-1} c_j (g\omega^i)^j$

```
#define MONT 2285 // 2^16 mod q
#define KYBER_ROOT_OF_UNITY 17
#define KYBER_Q 3329
```
$=2^8 13+1$

- Iterative FFT 반복문 구성에 대한 의문 해결

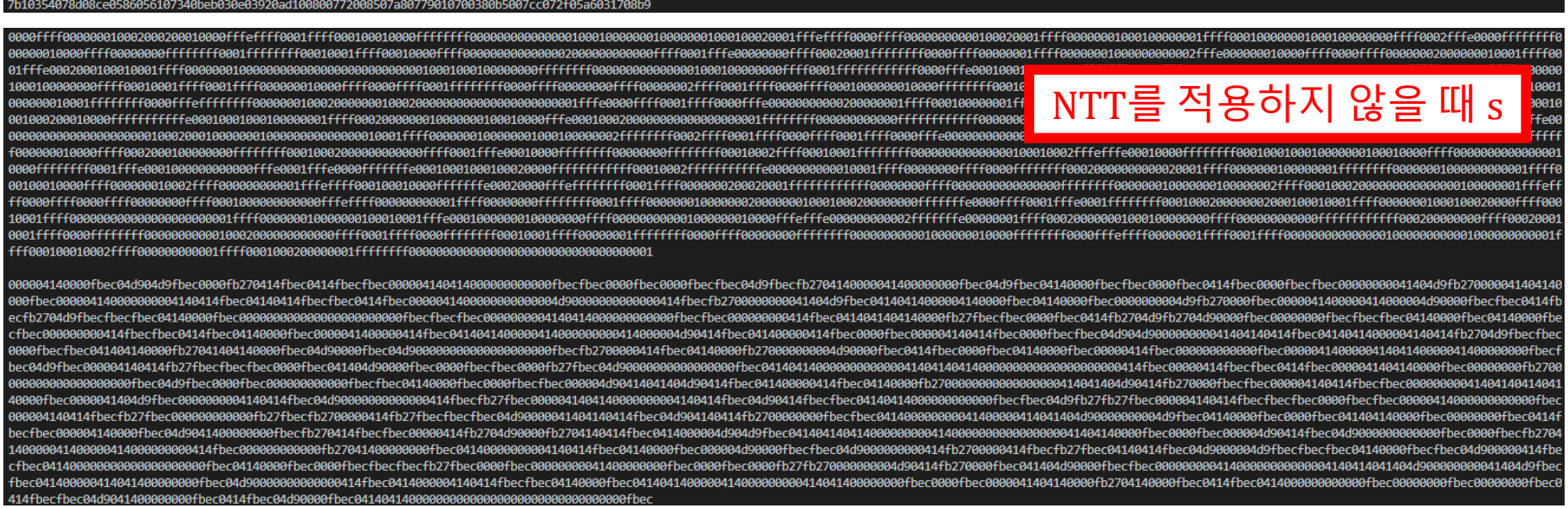
Algorithm 1 Forward NTT of a polynomial with precomputed roots of unity $\zeta_k = \zeta^{\text{brv}(k)}$

$k \leftarrow 1$
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;
      }
    }
  }
}
```

- Zeta 생성에 대한 분석
- NTT domain, Ring homorphism map 분석
- NTT를 사용하지 않고 다항식 곱셈을 수행할 때 키 공유가 가능한지 확인

- NTT를 사용하지 않고 다항식 곱셈을 수행할 때 키 공유가 가능한지 확인



감사합니다

Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography

Gregor Seiler*

IBM Research Zurich
grs@zurich.ibm.com

Abstract. Constant-time polynomial multiplication is one of the most time-consuming operations in many lattice-based cryptographic constructions. For schemes based on the hardness of Ring-LWE in power-of-two cyclotomic fields with completely splitting primes, the AVX2 optimized implementation of the Number-Theoretic Transform (NTT) from the NewHope key-exchange scheme is the state of the art for fast multiplication. It uses floating point vector instructions. We show that by using a modification of the Montgomery reduction algorithm that enables a fast approach with integer instructions, we can improve on the polynomial multiplication speeds of NewHope and Kyber by a factor of 4.2 and 6.3 on Skylake, respectively.

Keywords: lattice cryptography · NTT · implementation · AVX

1 Introduction

Lattice-based cryptography has emerged as a promising candidate for public-key cryptography that is still secure after the likely advent of quantum computers.

From a computational point of view, many lattice-based cryptographic schemes are based on operations in polynomial rings of the form $(\mathbb{Z}/q\mathbb{Z})[X]/(f(x))$ where f is an irreducible polynomial over $\mathbb{Z}$ and q is a prime number. For schemes whose security is based on the (presumed) hardness of the Ring-LWE problem, f is usually chosen to be a power-of-two cyclotomic $X^n + 1$ whose roots have order $2n$ and q is a prime that splits completely (in the number field). The latter condition is equivalent to $q \equiv 1 \pmod{2n}$. The reason for power-of-two cyclotomic rings and fully splitting primes is that the splitting behaviour of such primes in these rings allows for fast multiplication using the Fast Fourier Transform (FFT), which is also called the Number-Theoretic Transform (NTT) if it is performed over the base field $\mathbb{Z}_q$. A full NTT-based multiplication of two polynomials needs two forward NTTs to transform the input polynomials, a cheap pointwise vector multiplication and one inverse NTT. An advantage of NTT-based multiplication over other multiplication methods is that one can often save NTTs by sampling polynomials directly in the NTT domain, by storing the NTT domain representation of polynomials for later use, or by making use of the linearity of the NTT when computing sums of products of polynomials. This leads to important speed-ups as multiplication is frequently the most time consuming single operation in lattice-based schemes.

Producing fast constant-time implementations of the NTT for power-of-two cyclotomic fields has received considerable attention in cryptography during the last couple of years [GOPS13, ADPS16, GS16b, LN16]. For optimal speed on current Intel processors one has to exploit the fact that the NTT is easily vectorizable and needs to make use of

*Gregor Seiler was supported by the SNSF ERC Transfer Starting Grant CRETP2-166734-FELICITY and the H2020 Project SAFEcrypto

The fact that q is a prime number such that $q \equiv 1 \pmod{2n}$ means that $2n$ divides the order $q-1$ of the cyclic group $\mathbb{Z}_q^\times$. So $\mathbb{Z}_q$ contains $n = \varphi(2n)$ primitive $2n$-th roots of unity ζ^i where $i = 1, 3, \ldots, 2n-1$. It follows that $X^n + 1$ factors into linear polynomials $X - \zeta^i$ over $\mathbb{Z}_q$. Now recall that the Chinese remainder theorem says the natural ring homomorphism

$$f \mapsto \left(f(\zeta), f\left(\zeta^3\right), \ldots, f\left(\zeta^{2n-1}\right)\right) : \mathbb{Z}_q[X]/(X^n+1) \to \prod_i \mathbb{Z}_q[X]/(X-\zeta^i)$$

is in fact an isomorphism. The NTT computes this isomorphism and we write $\mathrm{NTT} : R_q \to \mathbb{Z}_q^n$ for it. Then the product fg of two polynomials $f, g \in R_q$ can be computed as $fg = \mathrm{NTT}^{-1}(\mathrm{NTT}(f)\,\mathrm{NTT}(g))$ which involves two forward NTTs one inverse NTT and the pointwise multiplication in $\mathbb{Z}_q^n$. Sometimes one can save NTTs. For example,

$$\sum_{i=1}^{t} f_i g_i = \sum_{i=1}^{t} \mathrm{NTT}^{-1}(\mathrm{NTT}(f_i)\,\mathrm{NTT}(g_i)) = \mathrm{NTT}^{-1}\left(\sum_{i=1}^{t} \mathrm{NTT}(f_i)\,\mathrm{NTT}(g_i)\right).$$

We recall the FFT-trick. See the excellent survey [Ber01]. The Fast Fourier Transform is the observation that isomorphisms such as the one above can be computed quickly in a divide and conquer fashion. Concretely, as $X^n + 1 = X^n - \zeta^n$, one can first compute the Chinese remainder map

$$f \mapsto \left(f \bmod X^{n/2} - \zeta^{n/2}, f \bmod X^{n/2} + \zeta^{n/2}\right)$$
$$\mathbb{Z}_q[X]/(X^n - \zeta^n) \to \mathbb{Z}_q[X]/(X^{n/2} - \zeta^{n/2}) \times \mathbb{Z}_q[X]/(X^{n/2} + \zeta^{n/2})$$

A slight modification of the standard FFT from above is the following. We have the twisting isomorphism

$$X \mapsto \zeta X : \mathbb{Z}_q[X]/(X^n + 1) \to \mathbb{Z}_q[X]/(X^n - 1).$$

Then the CRT map followed by twisting the second factor yields the map

$$\mathbb{Z}_q[X]/(X^n - 1) \to \mathbb{Z}_q[X]/(X^{\frac{n}{2}} - 1) \times \mathbb{Z}_q[X]/(X^{\frac{n}{2}} + 1) \to \mathbb{Z}_q[X]/(X^{\frac{n}{2}} - 1) \times \mathbb{Z}_q[X]/(X^{\frac{n}{2}} - 1)$$