Cryptography Optimization and Application LAb

NTT: Number Theoritic Transform 소개

2021.07.13

국민대학교 금융정보보안학과
서 석 충

목차

- 다항식 링 곱셈
- FFT/NTT 기본
- NTT 응용
- PQC에서 NTT 적용 사례

다항식 링 (Polynomial Ring)

- Ring (환)
  - (R, +)
    - (덧셈 결합 법칙) 임의의 r, s, t$\in$ R 에 대해, (r+s)+t = r+(s+t)
    - (덧셈 교환 법칙) 임의의 r, s$\in$ R 에 대해, r+s = s+r
    - (덧셈 항등원 존재) 임의의 r$\in$ R 에 대해, 0_R+r = r인 원소 $0_R \in$R이 존재
    - (덧셈 역원 존재) 임의의 r$\in$ R 에 대해, r+(-r) = 0_R인 원소 –r $\in$R이 존재
  - (R, ·)
    - (곱셈 결합 법칙) 임의의 r, s, t$\in$ R 에 대해, (rs)t = r(st)
    - (곱셈 항등원 존재) 임의의 r$\in$ R 에 대해, r1_R = r인 원소 $1_R \in$R이 존재
    - (덧셈과 곱셈 사이에 분배 법칙 성립) 임의의 r$\in$ R 에 대해, r(s+t)=rs+rt
      ※ 곱셈에서의 교환법칙은 성립 안 함
  - Ring의 예
    - Z (정수), Q (유리수), R (실수), C (복소수), Z/nZ (잉여환: Z를 n으로 나눈 나머지), 다항식 링(R[t])

다항식 링 (Polynomial Ring)

- 다항식 (Polynomial)
  - 다항식은 여러 개의 항으로 구성된 식 (계수와 차수의 곱을 모두 더한 것)

$$F(X) = a_0 + a_1 X + \cdots + a_{m-1} X^{m-1} + a_m X^m$$

$$F(X) = \sum_{k=0}^{m} a_k X^k$$

- 다항식 환 (Polynomial Ring)
  - 계수가 Ring인 다항식 환

$$F(X) = a_0 + a_1 X + a_2 X^2 + \ldots + a_n X^n = \sum_{i=0}^{n} a_i X^i, \text{ where } a_i \in R \text{ and } 0 \leq i \leq n$$

- 몫환 (Quotient Ring)
  - 최대 차수가 n이고, R_q 에서의 덧셈, 곱셈 연산은 다시 R_q의 원소가 되어야 함 (감산 다항식 필요)

$$R_q = \mathbb{Z}_q[X]/(X^n + 1)$$

암호에서의 다항식 연산

- 타원곡선암호 (이진 확장체 상에서의 타원곡선) → 소수 확장체로도 확장 가능

$$\mathbb{F}_{2^m} = \{a_{m-1}z^{m-1} + a_{m-2}z^{m-2} + \cdots + a_2z^2 + a_1z + a_0 \ : \ a_i \in \{0, 1\}\}$$

0	z^2	z^3	z^3+z^2
1	z^2+1	z^3+1	z^3+z^2+1
z	z^2+z	z^3+z	z^3+z^2+z
$z+1$	z^2+z+1	z^3+z+1	z^3+z^2+z+1

✓ $F_2{}^4$의 원소(기약다항식: $f(z)=z^4+z+1$)

Multiplication: $(z^3+z^2+1)\cdot(z^2+z+1) = z^2+1$ since

$$(z^3+z^2+1)\cdot(z^2+z+1) = z^5+z+1$$

and

$$(z^5+z+1) \bmod (z^4+z+1) = z^2+1.$$

✓ $F_2{}^4$의 원소에 대한 곱셈 연산

암호에서의 다항식 연산

- 양자내성암호
  - 대부분의 격자 기반 암호에서 메인 연산이 다항식 (링) 곱셈 연산임
  - Saber $\mathbb{Z}_q[X]/(X^n+1)$, q = 2^{13}
  - Kyber, Dilithium $\mathbb{Z}[X]/(X^n+1)$
    - Kyber: q = 3329
    - Dilithium: q = 8380417
  - NTRU $\mathbb{Z}[\mathbf{x}]/(\mathbf{\Phi}_1\mathbf{\Phi}_n)$ $\mathbb{Z}[\mathbf{x}]/(\mathbf{\Phi}_n)$

    $\mathbf{\Phi}_1$ is the polynomial $(\mathbf{x}-1)$
    $\mathbf{\Phi}_n$ is the polynomial $(\mathbf{x}^n-1)/(\mathbf{x}-1) = \mathbf{x}^{n-1}+\mathbf{x}^{n-2}+\cdots+1$
  - Falcon $f, g, F, G \in \mathbb{Z}[x]/(\phi)$ $\phi = x^n + 1$ for $n = 2^\kappa$

다항식 곱셈

- 격자기반 PQC에서의 NTT를 적용하기 위해 Ring은 $Z_q[X]/(X^n+1)$로 정의됨 (n = 2^k^)
  - 다항식의 계수는 modulo q 상에서 존재함
  - 다항식의 차수는 modulo (X^n+1) 상에서 존재함
- 다항식 곱셈 ($O(n^2)$)
  - $A(X), B(X) \in Z_q[X]/(X^n+1)$에 대하여, $A(X) \cdot B(X)\ mod\ X^n+1$

$$A(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_{n-1}x^{n-1},\ B(x) = b_0 + b_1x + b_2x^2 + b_3x^3 + \cdots + b_{n-1}x^{n-1}$$

$$
\begin{array}{rr}
 & (a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_{n-1}x^{n-1}) \cdot b_0 \\
 & (a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_{n-1}x^{n-1}) \cdot b_1x \\
 & (a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_{n-1}x^{n-1}) \cdot b_2x^2 \\
 & \cdots \\
+ & (a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_{n-1}x^{n-1}) \cdot b_{n-1}x^{n-1} \\
\hline
\end{array}
$$

FFT 기반 다항식 곱셈

- FFT (Fast Fourier Transform)
  - 이산 푸리에 변환과 그 역변환을 빠르게 수행하는 알고리즘
  - 디지털 신호 처리에 활용됨

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N} \qquad k = 0, \dots, N-1,$$

where $e^{i2\pi/N}$ is a primitive Nth root of 1

- FFT 기반 다항식 곱셈 아이디어

1. 다항식 $A(x) = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1}$ 를 나타내는 방법
   - Coefficient representation : $A(x) = (a_0, \dots., a_{n-1})$
   - Point-value representation
     - $n-1$ 차 다항식은 서로 다른 n 개의 점으로 유일하게 표현할 수 있다 (Lagrange's Interpolation)
       - n number of points $(x_i, y_i), 0 \le i \le n-1$ such that
       - $x_i \ne x_j$ for $0 \le i \ne j \le n-1$
       - $y_i = A(x_i)$ for $0 \le i \le n-1$

FFT 기반 다항식 곱셈

- FFT 기반 다항식 곱셈 아이디어

  2. 두 $n-1$ 차 다항식 $A(x), B(x)$ 의 곱 $C(x) = A(x)B(x)$

     - $C(x_i) = A(x_i)B(x_i) = y_i y_i'$
     - $A(1) = 3, B(1) = 2$ 라 가정하면 $C(1) = 3 \times 2$

➡ **$C(x)$ 는 $2n-2$ 차 ➔ $2n-2$ 개의 점으로 표현 가능**

➡ **만약 $2n-2$ 개의 점에 대해 $\left(x_i, A(x_i)\right), (x_i, B(x_i))$ 를 구했다면?**

➡ **$C(x) \Leftrightarrow \left(x_i, A(x_i)B(x_i)\right) = (x_i, C(x_i))$**

FFT 기반 다항식 곱셈

- FFT 기반 다항식 곱셈 아이디어
  - Example)
    - $A(x) = x^2 - 1, B(x) = x + 1$
    - $A(x)B(x)$ ➔ 3차 ➔ 4 coefficients

$\boldsymbol{A(-1) = 0}$
$\boldsymbol{A(1) = 0}$
$\boldsymbol{A(2) = 3}$
$\boldsymbol{A(3) = 8}$

$\boldsymbol{B(-1) = 0}$
$\boldsymbol{B(1) = 2}$
$\boldsymbol{B(2) = 3}$
$\boldsymbol{B(3) = 4}$

$\boldsymbol{C(-1) = A(-1)B(-1) = 0}$
$\boldsymbol{C(1) = A(1)B(1) = 0}$
$\boldsymbol{C(2) = A(2)B(2) = 9}$
$\boldsymbol{C(3) = A(3)b(3) = 24}$

$\boldsymbol{C(x) = x^3 + x^2 - x - 1}$

FFT 기반 다항식 곱셈

- FFT 기반 다항식 곱셈 아이디어

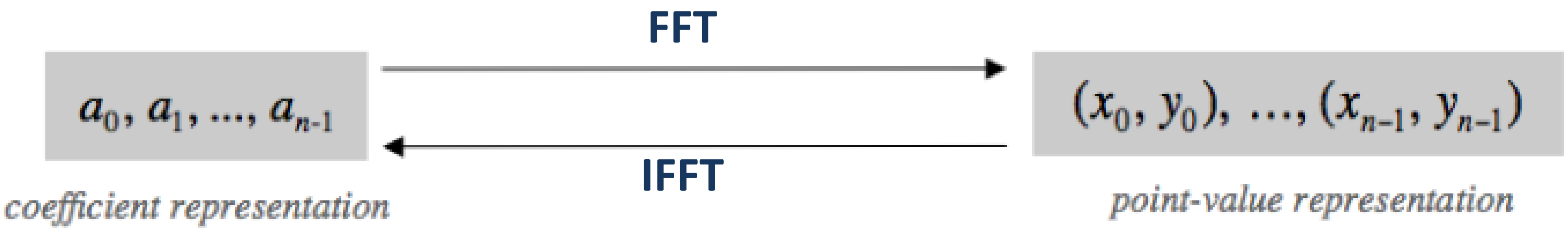
  3. 임의의 point를 이용해서 함수값을 계산하는 대신, $n-th$ root of unity 값을 이용

     즉, FFT는 함수 $f(x)$ 에 대해 $n-th$ root of unity 에 대한 evaluation ($\omega \in \mathrm{C}$)

$$\boldsymbol{f(\omega^0), f(\omega^1), \dots, f(\omega^{n-1})}$$

✓ CRT를 적용하는 것과 유사함

- $\omega^n = 1, \omega^{\frac{n}{2}+k} = -\omega^k$ 등의 성질을 이용하면 divide and conquer 가능



FFT 기반 다항식 곱셈

- FFT 기반 다항식 곱셈 아이디어
  - Example)
    - $A(x) = a_0 + a_1x + a_2x^2 + a_3x^3$
    - $\omega^4 = 1$ (4-th root of unity) ➔ $\boldsymbol{\omega^0 = 1, \omega = i}, \omega^2 = -1, \omega^3 = -i$
      - Split polynomial : $A(x) = (a_0 + a_2x^2) + (a_1x + a_3x^3) = \underbrace{(a_0 + a_2x^2)}_{\boldsymbol{A_0(x^2)}} + \underbrace{x(a_1 + a_3x^2)}_{\boldsymbol{xA_1(x^2)}}$

        Aeven xAodd

    - $A(w^0) = A(1) = A_0(1) + 1 \cdot A_1(1)$
    - $A(w^1) = A(i) = A_0(-1) + i \cdot A_1(-1)$
    - $A(w^2) = A(-1) = A_0(1) - 1 \cdot A_1(1)$
    - $A(w^3) = A(-i) = A_0(-1) - iA_1(-1)$

$\boldsymbol{A_0(1), A_1(1)}$ **값 ➔** $\boldsymbol{A(\omega^0), A(\omega^2)}$ **연산 가능**

FFT 기반 다항식 곱셈

- FFT 기반 다항식 곱셈 아이디어
  - Example)
    - $A(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7$
    - $\omega^8 = 1$ (8-th root of unity) ➔ $\boldsymbol{\omega^0 = -\omega^4, \omega^1 = -\omega^5, \omega^2 = -\omega^6, \omega^3 = -\omega^7}$
      - Split polynomial : $A(x) = (a_0+a_2x^2 + a_4x^4 + a_6x^6) + (a_1x + a_3x^3 + a_5x^5 + a_7x^7)$

$$= (\underbrace{a_0+a_2x^2 + a_4x^4 + a_6x^6}_{\boldsymbol{A_0(x^2)}}) + x(\underbrace{a_1 + a_3x^2 + a_5x^4 + a_7x^6}_{\boldsymbol{xA_1(x^2)}})$$

Aeven xAodd

- $A(1) = A_0(1) + 1 \cdot A_1(1)$
- $A(\omega) = A_0(\omega^2) + \omega \cdot A_1(\omega^2)$
- $A(\omega^2) = A_0(\omega^4) + \omega^2 A_1(\omega^4)$
- $A(\omega^3) = A_0(\omega^6) + \omega^3 A_1(\omega^6)$
- $A(\omega^4) = A_0(\omega^8) + \omega^4 A_1(\omega^8) \quad = A_0(1) - A_1(1)$
- $A(\omega^5) = A_0(\omega^{10}) + \omega^5 A_1(\omega^{10}) = A_0(\omega^2) - \omega A_1(\omega^2)$
- $A(\omega^6) = A_0(\omega^{12}) + \omega^6 A_1(\omega^{12}) = A_0(\omega^4) - \omega^2 A_1(\omega^4)$
- $A(\omega^7) = A_0(\omega^{14}) + \omega^7 A_1(\omega^{14}) = A_0(\omega^6) - \omega^3 A_1(\omega^6)$

✓ 동일한 값을 곱한 후에 덧셈 또는 뺄셈 수행

→ Butterfly 방법

✓ 이 과정을 recursive하게 반복함(최대한 낮은 차수까지)

NTT 기반 다항식 곱셈

- Number Theoretic Transform
  - 조건: Complex number에서 정수 도메인으로 이동
    - 적절한 modulus q 를 선택해 n –th root of unity 가 $\mathbb{Z}_q$ 에 존재하도록
    - $\mathbf{q \equiv 1 \bmod 2n}$
      - $\boldsymbol{x^{2n}} \equiv \mathbf{1 \bmod q}$ 를 만족하는 모든 정수가 Z_q 안에 있음 (primitive 2n-th root of unity)
  - 연산량: $O(n^2) \rightarrow O(n\log_2 n)$
  - 절차: NTT 변환 ($O(n\log_2 n)$) → pointwise 곱셈 ($O(n)$) → INVNTT 변환 ($O(n\log_2 n)$)

NTT 기반 다항식 곱셈

- CRT (Chinese Remainder Theorem)
  - 큰수에 대한 연산을 작은 수로 분할하여 처리할 수 있는 방법 (RSA 지수승 연산에서 사용됨)
  - 특징
    - $M = m_1 \cdots m_k$일 경우 (m_i coprime), k개의 ($a_i \; mod \; m_i$)는 mod M상에서의 유일한 정수 x 를 결정함
  - 예) $N = 11 \cdot 23 = 253$ 일때, 다음을 만족하는 x를 찾아라

$$x \equiv 3 \bmod 11$$
$$x \equiv 21 \bmod 23$$

$0 \cdot 23 + 21 = 21$	$\stackrel{?}{\equiv} 3 \bmod 11$	Nope.
$1 \cdot 23 + 21 = 44$	$\stackrel{?}{\equiv} 3 \bmod 11$	Nope.
$2 \cdot 23 + 21 = 67$	$\stackrel{?}{\equiv} 3 \bmod 11$	Nope.
$3 \cdot 23 + 21 = 90$	$\stackrel{?}{\equiv} 3 \bmod 11$	Nope.
$4 \cdot 23 + 21 = 113$	$\stackrel{?}{\equiv} 3 \bmod 11$	**Yes!**

$$x \equiv \sum_1^k a_i \cdot M_i \cdot x_i \text{ mod M}$$

$$M_i = M/m_i,\ N_i x_i \equiv 1 \bmod m_i$$

$$(3 \cdot 23 \cdot 1 + 21 \cdot 11 \cdot 21 = 4920) \; mod \; 253$$
$$\rightarrow 113$$

NTT 기반 다항식 곱셈

- CRT (Chinese Remainder Theorem)
  - $a_i \ mod \ m_i$에 대한 연산이 x에 대해서도 성립함 (Residue Number System)
    - 예) $k \cdot a_i \ mod \ m_i \ \rightarrow \ kx \ mod \ M$

$$x \equiv 3 \cdot 3 \quad \equiv 9 \bmod 11$$
$$x \equiv 3 \cdot 21 \equiv 17 \bmod 23$$

$$\rightarrow (9 \cdot 23 \cdot 1 + 17 \cdot 11 \cdot 21 = 4134) \equiv 86 \ mod \ 253$$
$$\rightarrow \ 3 \cdot 113 = 339 \equiv 86 \ mod \ 253$$

- 다항식 분할 (Toy example)
  - 숫자 분할에서 다항식 링 분할(감산 다항식)
  - m_i는 모두 1보다 커야 하며, coprime이어야 함
  - $q \equiv 1 \; mod \; 2n$ 만족 해야함
    - $17 \equiv 1 \; mod \; 8$

$$R = \mathbb{Z}_q[X]/(X^n+1) = \mathbb{Z}_{17}[X]/(X^4+1)$$

$$m_1 = (X^2 - 4)$$
$$m_2 = (X^2 + 4)$$

$$\begin{aligned} m_1 m_2 &= (X^2-4)(X^2+4) \\ &= X^4 - \cancel{4X^2} + \cancel{4X^2} - 16 \\ &= X^4 - 16 \\ &\equiv X^4 + 1 \quad \checkmark \end{aligned}$$

NTT 기반 다항식 곱셈

- 다항식 분할 (Toy example)
  - Ring을 subring의 곱셈으로 표현(서로소인 subring의 곱으로 표현)
  - 다항식 a를 subring 상의 원소로 표현

$$R = \mathbb{Z}_q[X]/(X^n + 1) = \mathbb{Z}_{17}[X]/(X^4 + 1)$$

$$R_1 = \mathbb{Z}_{17}[X]/(X^2 - 4) \quad R_2 = \mathbb{Z}_{17}[X]/(X^2 + 4)$$

$$\begin{aligned} \mathbf{a} \bmod (X^2 - 4) &\equiv 2 + 7X^3 \\ &\equiv 2 + 7X^3 - 7X(X^2 - 4) \\ &\equiv 2 + (7 \cdot 4)X \\ &\equiv 2 + 28X \\ &\equiv 2 + 11X \end{aligned}$$

$$\begin{aligned} \mathbf{a} \bmod (X^2 + 4) &\equiv 2 + 7X^3 \\ &\equiv 2 + 7X^3 - 7X(X^2 + 4) \\ &\equiv 2 - (7 \cdot 4)X \\ &\equiv 2 - 28X \\ &\equiv 2 + 6X \end{aligned}$$

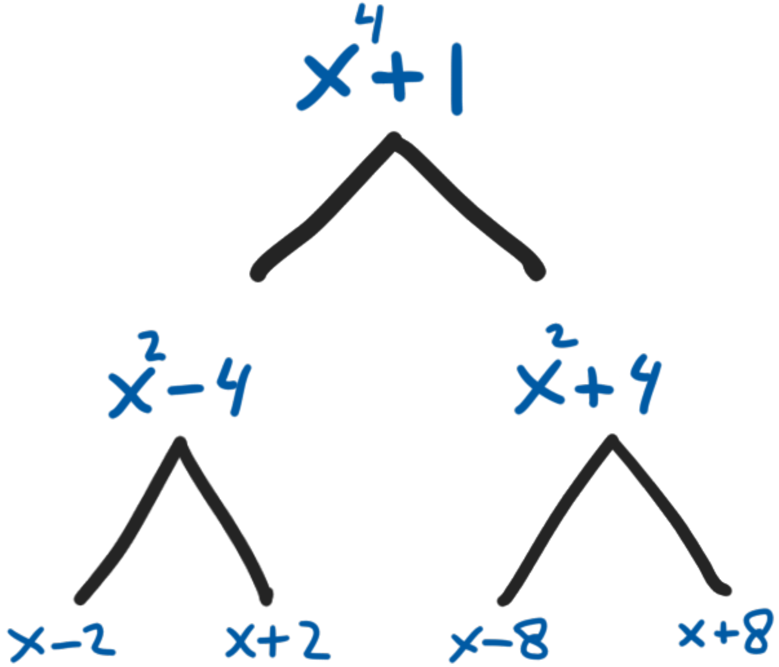
NTT 기반 다항식 곱셈

- 다항식 분할 (Toy example)
  - 다항식이 0차로 까지 표현될 수 있도록 Ring을 subring으로 재귀적으로 분할

1. $\mathbb{Z}_{17}[X]/(X^4+1)$,
   $(X^4+1) = (X^2-\zeta_1)(X^2+\zeta_1)$
   $\rightarrow \zeta_1^2 = -1 \bmod 17, \rightarrow \zeta_1 = 4$

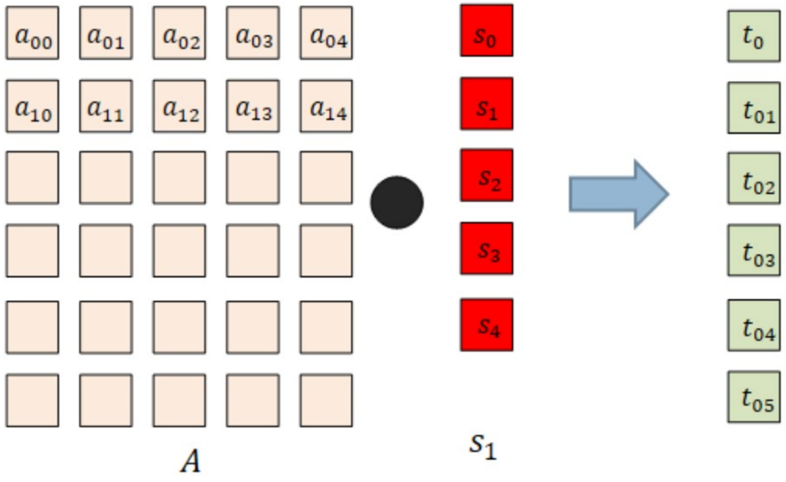
2. $\mathbb{Z}_{17}[X]/(X^2-4)$,
   $(X^2-4) = (X^2-\zeta_1) = (X-\zeta_2)(X+\zeta_2)$
   $\rightarrow \zeta_2^2 = 4 \bmod 17, \rightarrow \zeta_2 = 2$

3. $\mathbb{Z}_{17}[X]/(X^2+4)$,
   $(X^2+4) = (X^2+\zeta_1) = (X-\zeta_3)(X+\zeta_3)$
   $\rightarrow \zeta_3^2 = -4 \bmod 17, \rightarrow \zeta_3 = 8$



NTT 기반 다항식 곱셈

- CRT 개념을 적용하여 NTT 기반 다항식 곱셈 수행
  1. 다항식 a와 b를 n개의 0차 다항식으로 분할 (NTT 변환)
  2. Pointwise 곱셈 수행
  3. INVNTT 변환을 통해 곱셈 결과 c 복원
- Dilithium에서의 다항식 곱셈
  - n=256, q = 8380417 ($q \equiv 1$ mod $2n$)



Gen

$\mathbf{A} \leftarrow R_q^{k\times\ell}$

$(\mathbf{s}_1, \mathbf{s}_2) \leftarrow S_\eta^\ell \times S_\eta^k$

$\boxed{\mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2}$

return $(pk = (\mathbf{A}, \mathbf{t}), sk = (\mathbf{A}, \mathbf{t}, \mathbf{s}_1, \mathbf{s}_2))$

Sign(sk, M)

$\mathbf{z} := \perp$

while $\mathbf{z} = \perp$ do

$\quad \mathbf{y} \leftarrow S_{\gamma_1-1}^\ell$

$\quad \mathbf{w}_1 := \boxed{\mathsf{HighBits}(\mathbf{A}\mathbf{y}, 2\gamma_2)}$

$\quad c \in B_\tau := \mathsf{H}(M \,\|\, \mathbf{w}_1)$

$\quad \boxed{\mathbf{z} := \mathbf{y} + c\mathbf{s}_1}$

$\quad$ **if** $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or $\|\mathsf{LowBits}(\mathbf{A}\mathbf{y} - c\mathbf{s}_2, 2\gamma_2)\|_\infty \geq \gamma_2 - \beta$, then $\mathbf{z} := \perp$

return $\sigma = (\mathbf{z}, c)$

Verify$(pk, M, \sigma = (\mathbf{z}, c))$

$\mathbf{w}_1' := \boxed{\mathsf{HighBits}(\mathbf{A}\mathbf{z} - c\mathbf{t}, 2\gamma_2)}$

if return $[\![\|\mathbf{z}\|_\infty < \gamma_1 - \beta]\!]$ **and** $[\![c = \mathsf{H}(M \,\|\, \mathbf{w}_1')]\!]$

[그림 11] Crystals-Dilithium의 KeyGen, Sign, Verify 과정*

NTT 기반 다항식 곱셈

- Dilithium에서의 다항식 곱셈 (다항식 분할)
  - n=256, q = 8380417 ($q \equiv 1 \; mod \; 2n$)

Compute $c = a \cdot b \bmod \mathbb{R}_q$ where $a, b \in \mathbb{R}_q = \mathbb{Z}_q[X]/(X^{256}+1)$

1. $\mathbb{R}_q = \mathbb{Z}_q[X]/(X^{256}+1)$,
   $(X^{256}+1) = (X^{128}-\alpha_1)(X^{128}+\alpha_1)$,
   $\rightarrow 1 = -\alpha_1^2 \rightarrow \alpha_1^4 = 1 \rightarrow \alpha_1 = \zeta_4$

2. $(X^{128}-\alpha_1) = (X^{64}-\alpha_2)(X^{64}+\alpha_2)$,
   $\rightarrow \alpha_1 = \alpha_2^2 \rightarrow \alpha_2 = \sqrt{\alpha_1} = \alpha_1^{\frac{1}{2}} \rightarrow \alpha_2 = \zeta_8$

3. $(X^{128}+\alpha_1) = (X^{64}-\alpha_3)(X^{64}+\alpha_3)$,
   $\rightarrow \alpha_1 = -\alpha_3^2 \rightarrow \alpha_3 = \sqrt{-\alpha_1} = \sqrt{(-1)\alpha_1} = \sqrt{\alpha_1^2 \cdot \alpha_1} = \alpha_1^{\frac{3}{2}} \rightarrow \alpha_3 = \zeta_8^3$

$$(X^{256}+1) = (X^{64}-\zeta_8)(X^{64}+\zeta_8)(X^{64}-\zeta_8^3)(X^{64}+\zeta_8^3)$$

NTT 기반 다항식 곱셈

- Dilithium에서의 다항식 곱셈 (다항식 분할)
  - n=256, $q = 8380417$ $(q \equiv 1 \; mod \; 2n)$

$$(X^{256}+1) = (X^{64}-\zeta_8)(X^{64}+\zeta_8)(X^{64}-\zeta_8^3)(X^{64}+\zeta_8^3)$$

1. $(X^{64}-\zeta_8) = (X^{32}-\alpha_4)(X^{32}+\alpha_4),$
   $\rightarrow \zeta_8 = \alpha_4^2 \rightarrow \alpha_4 = \zeta_{16}$

2. $(X^{64}+\zeta_8) = (X^{32}-\alpha_5)(X^{32}+\alpha_5),$
   $\rightarrow \zeta_8 = -\alpha_5^2 \rightarrow \alpha_5^2 = -\zeta_8 \rightarrow \alpha_5 = \sqrt{(-1)\zeta_8} = \sqrt{\alpha_1^2 \cdot \zeta_8} =$
   $\sqrt{\zeta_4^2 \cdot \zeta_8} = \sqrt{\zeta_8^4 \cdot \zeta_8} = \sqrt{\zeta_8^5} = \zeta_{16}^5$

$$\boxed{(X^{32}-\zeta_{16})(X^{32}+\zeta_{16})(X^{32}-\zeta_{16}^5)(X^{32}+\zeta_{16}^5)}(X^{32}-\zeta_{16}^3)(X^{32}+\zeta_{16}^3)(X^{32}-\zeta_{16}^7)(X^{32}+\zeta_{16}^7)$$

NTT 기반 다항식 곱셈

- Dilithium에서의 다항식 곱셈 (다항식 분할)
  - n=256, $q = 8380417$ ($q \equiv 1 \; mod \; 2n$)

$$(X^{256}+1) =$$

$$(X-\zeta)(X+\zeta)(X-\zeta^{129})\cdots(X-\zeta^{127})(X+\zeta^{127})(X-\zeta^{255})(X+\zeta^{255})$$

$$\text{where } \zeta = \zeta_{512}$$

NTT 기반 다항식 곱셈

- Dilithium에서의 다항식 곱셈 (원소 표현)
  - Ring이 256개의 서로소인 subring으로 분할되는 것을 확인
    → 256개의 subring의 원소로 다항식을 표현

$$\mathbf{a} \in \mathbb{Z}_q[X]/(X^{256}+1)$$
$$\rightarrow \mathbf{a}_L^{(1)} \in \mathbb{Z}_q[X]/(X^{128}-\zeta^{128}),\ \mathbf{a}_R^{(1)} \in \mathbb{Z}_q[X]/(X^{128}+\zeta^{128})$$

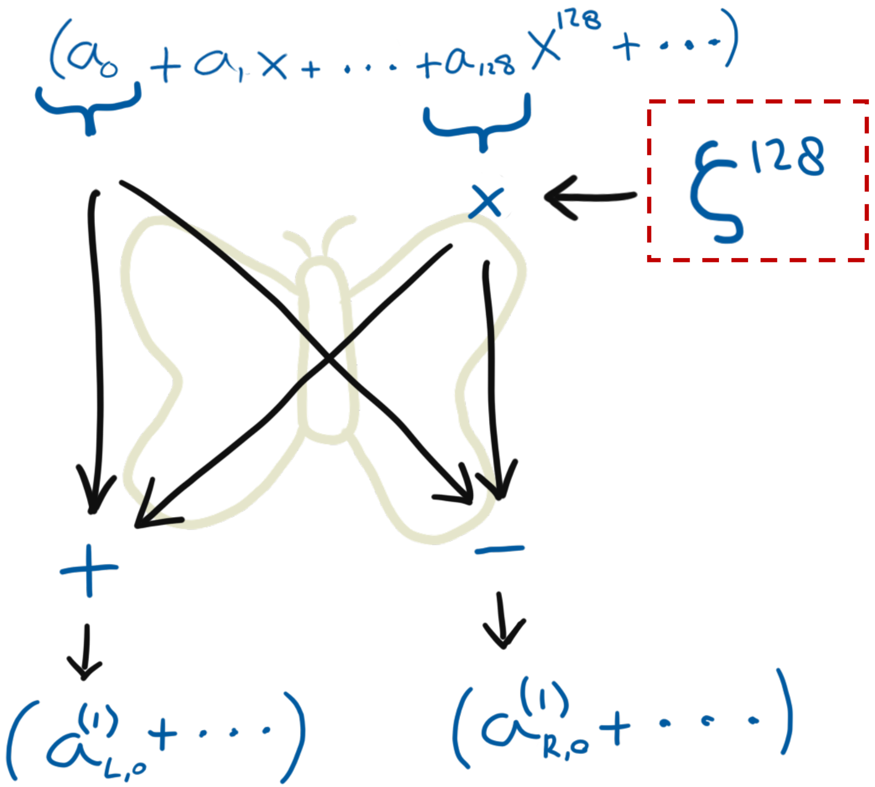
X^{255}									X^{128}	X^{127}									X^{0}
a_{255}									a_{128}	a_{127}									a_0

$$\mathbf{a}_L^{(1)} = (a_0+\zeta^{128}a_{128}) + (a_1+\zeta^{128}a_{129})X + \cdots + (a_{127}+\zeta^{128}a_{255})X^{127},$$
$$\mathbf{a}_R^{(1)} = (a_0-\zeta^{128}a_{128}) + (a_1-\zeta^{128}a_{129})X + \cdots + (a_{127}-\zeta^{128}a_{255})X^{127}$$

NTT 기반 다항식 곱셈

- Dilithium에서의 다항식 곱셈 (원소 표현)

$$\mathbf{a}_L^{(1)} = (a_0 + \zeta^{128} a_{128}) + (a_1 + \zeta^{128} a_{129})X + \cdots + (a_{127} + \zeta^{128} a_{255})X^{127},$$
$$\mathbf{a}_R^{(1)} = (a_0 - \zeta^{128} a_{128}) + (a_1 - \zeta^{128} a_{129})X + \cdots + (a_{127} - \zeta^{128} a_{255})X^{127}$$



✓ ξ는 사전에 계산되어 저장된 값

✓ ξ와 곱하는 부분은 single-precision 곱셈

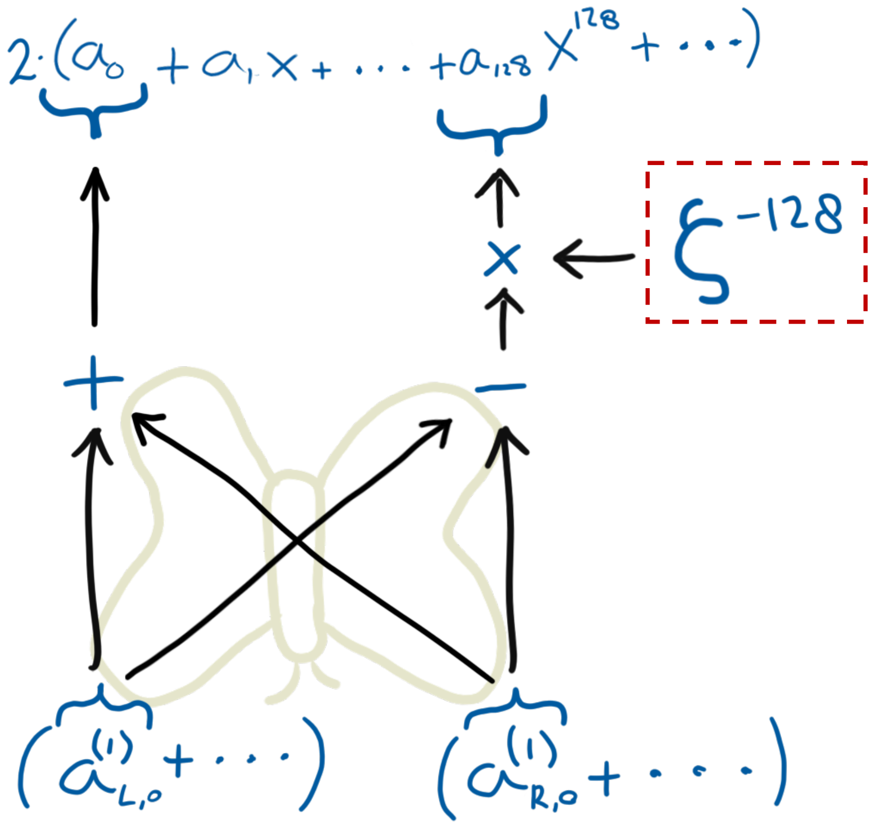
- Montgomery 기반 곱셈 적용: Mont(a_i, ξ) = $a_i \cdot \xi \cdot R^{-1}$ mod q
- ξ는 원래 값에 R^2을 곱하고 mod q를 한 값을 저장함 ($\xi R^2 \ mod \ q$)

✓ 이 과정을 $\log_2 n$만큼 반복함 (계층의 수)

→ 연산량: $O(n\log_2 n)$

NTT 기반 다항식 곱셈

- Dilithium에서의 다항식 곱셈 (역변환)



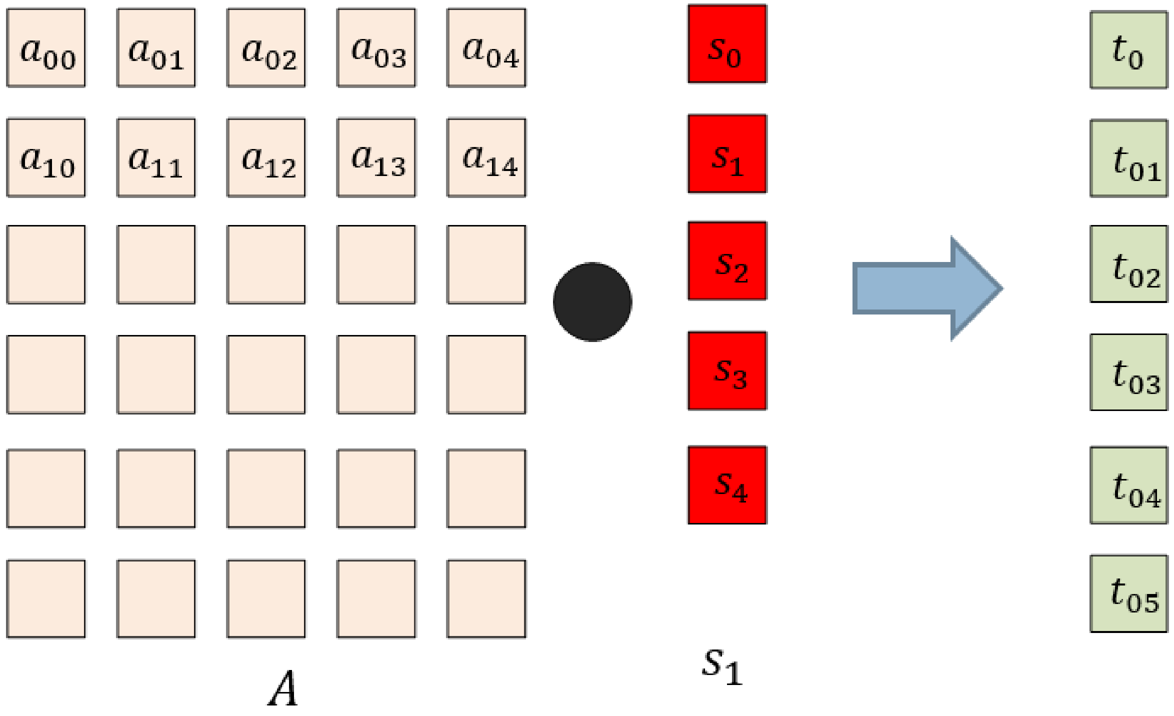
$$a^{(1)}_{L,0} = a_0 + \zeta^{128} a_{128}$$
$$a^{(1)}_{R,0} = a_0 - \zeta^{128} a_{128}$$

$$a^{(1)}_{L,0} + a^{(1)}_{R,0} = a_0 + \cancel{\zeta^{128} a_{128}} + a_0 - \cancel{\zeta^{128} a_{128}}$$
$$2a_0 = a^{(1)}_{L,0} + a^{(1)}_{R,0}$$
$$a_0 = 2^{-1}\left(a^{(1)}_{L,0} + a^{(1)}_{R,0}\right)$$

$$a^{(1)}_{L,0} - a^{(1)}_{R,0} = \cancel{a_0} + \zeta^{128} a_{128} - \cancel{a_0} + \zeta^{128} a_{128}$$
$$2\zeta^{128} a_{128} = a^{(1)}_{L,0} - a^{(1)}_{R,0}$$
$$a_{128} = 2^{-1}\zeta^{-128}\left(a^{(1)}_{L,0} - a^{(1)}_{R,0}\right)$$

NTT 기반 다항식 곱셈

- Dilithium 코드



$$t_i = \boxed{a_{i0} \times s_0} + \boxed{a_{i1} \times s_1} + \boxed{a_{i2} \times s_2} + \boxed{a_{i3} \times s_3} + \boxed{a_{i4} \times s_4}$$

256차 다항식 × 256차 다항식

```c
typedef struct {
  poly vec[L];
} polyvecl;
```

```c
typedef struct {
  int32_t coeffs[N];
} poly;
```

NTT 기반 다항식 곱셈

- Dilithium 코드

```c
/* Matrix-vector multiplication */
s1hat = s1;
polyvecl_ntt(&s1hat);
polyvec_matrix_pointwise_montgomery(&t1, mat, &s1hat);
polyveck_reduce(&t1);
polyveck_invntt_tomont(&t1);
```

```c
void polyvecl_ntt(polyvecl *v) {
  unsigned int i;

  for(i = 0; i < L; ++i)  경과 시간 1r
    poly_ntt(&v->vec[i]);
}
```

```c
void ntt(int32_t a[N]) {  경과 시간 1ms 이하
  unsigned int len, start, j, k;
  int32_t zeta, t;

  k = 0;
  for(len = 128; len > 0; len >>= 1) {
    for(start = 0; start < N; start = j + len) {
      zeta = zetas[++k];
      for(j = start; j < start + len; ++j) {
        t = montgomery_reduce((int64_t)zeta * a[j + len]);
        a[j + len] = a[j] - t;
        a[j] = a[j] + t;
      }
    }
  }
}
```

NTT 기반 다항식 곱셈

- Dilithium 코드

```
void invntt_tomont(int32_t a[N]) {
  unsigned int start, len, j, k;
  int32_t t, zeta;
  const int32_t f = 41978; // mont^2/256

  k = 256;
  for(len = 1; len < N; len <<= 1) {
    for(start = 0; start < N; start = j + len) {
      zeta = -zetas[--k];
      for(j = start; j < start + len; ++j) {
        t = a[j];
        a[j] = t + a[j + len];
        a[j + len] = t - a[j + len];
        a[j + len] = montgomery_reduce((int64_t)zeta * a[j + len]);
      }
    }
  }

  for(j = 0; j < N; ++j) {
    a[j] = montgomery_reduce((int64_t)f * a[j]);
  }
}
```

NTT 기반 다항식 곱셈

- Kyber
  - Dilithium과 기반 문제와 동작 방식이 매우 유사함
  - Kyber에서 사용하는 q = 3329로서, 512-th primitive root가 존재하지 않음
    - 256-th primitive root가 존재하므로, 1차까지만 다항식을 감산시킬 수 있음 (a_1X+a_0 형식)

- 다른 다항식 곱셈 알고리즘
  - Karatsuba 곱셈
  - Toom-Cook 곱셈
    - Saber에서 사용 사용하고 있음

$$C(x) = A(x) \times B(x) = \sum_{i=0}^{n-1} a_i x^i \times \sum_{i=0}^{n-1} b_i x^i$$

$$\begin{tikzcd}
Schoolbook \arrow[r] & Toom-Cook \arrow[r] & NTT \\
O(n^2) \arrow[u] & O(n^{\frac{\log(2k-1)}{\log k}}) \arrow[u] & O(n \log n) \arrow[u]
\end{tikzcd}$$

Q&A