

POST-QUANTUM CRYPTOGRAPHY

LECTURE 3: ADVANCED PQC

Florida Atlantic University

Fall 2025

1 Polynomial Multiplication

- Schoolbook Method
- NTT
- Butterfly Algorithms
- Full NTT Algorithm
- Incomplete NTT Algorithm

2 ML-KEM

- ML-KEM Components
- Full ML-KEM Algorithm
- ML-KEM Parameter Sets

3 Notes on ML-DSA

- ML-DSA Parameter Sets
- Expected Number of Iterations for Rejection Sampling

POLYNOMIAL MULTIPLICATION

POLYNOMIAL MULTIPLICATION: SCHOOLBOOK METHOD

✧ Computing the multiplication of two polynomials:

$$a(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}$$

$$b(x) = b_0 + b_1x + b_2x^2 + \cdots + b_{n-1}x^{n-1}$$

$$\Rightarrow c(x) = c_0 + c_1x + c_2x^2 + \cdots + c_{n-1}x^{n-1} = a(x) \cdot b(x) \pmod{\mathcal{R}_q}$$

✧ Traditional way (schoolbook):

1. $c(x) = a_0b_0 + a_0b_1x + a_0b_2x^2 + \cdots + a_1b_0x + a_1b_1x^2 + \cdots + a_{n-1}b_{n-1}x^{2n-2} \pmod{\mathcal{R}_q}$

2. Modulo $\mathcal{R}_q$:

- Replace x^{n+k} by $-x^k$.
- Reduce coefficients using Barret or Montgomery.

✧ $\mathcal{O}(n^2)$ operations: n^2 multiplications.

POLYNOMIAL MULTIPLICATION: SCHOOLBOOK METHOD

✧ Computing the multiplication of two polynomials:

$$a(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}$$

$$b(x) = b_0 + b_1x + b_2x^2 + \cdots + b_{n-1}x^{n-1}$$

$$\Rightarrow c(x) = c_0 + c_1x + c_2x^2 + \cdots + c_{n-1}x^{n-1} = a(x) \cdot b(x) \pmod{\mathcal{R}_q}$$

✧ Traditional way (schoolbook):

1. $c(x) = a_0b_0 + a_0b_1x + a_0b_2x^2 + \cdots + a_1b_0x + a_1b_1x^2 + \cdots + a_{n-1}b_{n-1}x^{2n-2} \pmod{\mathcal{R}_q}$

2. Modulo $\mathcal{R}_q$:

- Replace x^{n+k} by $-x^k$.
- Reduce coefficients using Barret or Montgomery.

✧ $\mathcal{O}(n^2)$ operations: n^2 multiplications.

EXAMPLE OF SCHOOLBOOK METHOD

$$\diamond a(x) = 2 + 8x + 4x^2 + x^3$$

$$\diamond b(x) = 5 + 6x + 7x^2 + 8x^3$$

$$\diamond \mathcal{R}_q = \mathbb{Z}_{11}[x]/(x^4 + 1)$$

$$\begin{array}{r}
 2 8x 4x^2 x^3 \\
 \times 10 2x 7x^2 9x^3 \\
 \hline
 18x^3 72x^4 36x^5 9x^6 \\
 14x^2 56x^3 28x^4 7x^5 \\
 4x 16x^2 8x^3 2x^4 \\
 + 20 80x 40x^2 10x^3 \\
 \hline
 20 84x 70x^2 92x^3 102x^4 43x^5 9x^6
 \end{array}$$

EXAMPLE OF SCHOOLBOOK METHOD

✧ Then, substitute $x^4 \equiv -1$, $x^5 \equiv -x$, $x^6 \equiv -x^2 \pmod{x^4 + 1}$:

$$\begin{aligned} & 20 + 84x + 70x^2 + 92x^3 + 102x^4 + 43x^5 + 9x^6 \\ &= 20 + 84x + 70x^2 + 92x^3 - 102 - 43x - 2x^2 \pmod{x^4 + 1} \\ &= -82 + 41x + 68x^2 + 92x^3 \pmod{x^4 + 1} \end{aligned}$$

✧ Then, using Barret or Montgomery reduction, reduce the coefficients modulo 11:

$$\begin{aligned} & -82 + 41x + 68x^2 + 92x^3 \\ &= 6 + 8x + 2x^2 + 4x^3 \pmod{11} \end{aligned}$$

✧ So, $a(x) \cdot b(x) \pmod{\mathcal{R}_q} = 6 + 8x + 2x^2 + 4x^3$.

POLYNOMIAL MULTIPLICATION: CONVOLUTION

✧ The discrete convolution of two vectors a and b with length n is defined as:

$$c_j = \sum_{i=0}^{n-1} a_i \cdot b_{j-i}, \quad j = \{0, 1, 2, \dots, 2n-2\}$$

✧ So, for example for $n = 4$ and $a = [a_0, a_1, a_2, a_3]$ and $b = [b_0, b_1, b_2, b_3]$:

$$c_0 = a_0 \cdot b_0$$

$$c_1 = a_0 \cdot b_1 + a_1 \cdot b_0$$

$$c_2 = a_0 \cdot b_2 + a_1 \cdot b_1 + a_2 \cdot b_0$$

$$c_3 = a_0 \cdot b_3 + a_1 \cdot b_2 + a_2 \cdot b_1 + a_3 \cdot b_0$$

$$c_4 = a_1 \cdot b_3 + a_2 \cdot b_2 + a_3 \cdot b_1$$

$$c_5 = a_2 \cdot b_3 + a_3 \cdot b_2$$

$$c_6 = a_3 \cdot b_3$$

✧ It is exactly the same as the schoolbook multiplication (but without modulo $\mathcal{R}_q$) $\rightarrow \mathcal{O}(n^2)$.

POLYNOMIAL MULTIPLICATION: CONVOLUTION

✧ The discrete convolution of two vectors a and b with length n is defined as:

$$c_j = \sum_{i=0}^{n-1} a_i \cdot b_{j-i}, \quad j = \{0, 1, 2, \dots, 2n-2\}$$

✧ So, for example for $n = 4$ and $a = [a_0, a_1, a_2, a_3]$ and $b = [b_0, b_1, b_2, b_3]$:

$$c_0 = a_0 \cdot b_0$$

$$c_1 = a_0 \cdot b_1 + a_1 \cdot b_0$$

$$c_2 = a_0 \cdot b_2 + a_1 \cdot b_1 + a_2 \cdot b_0$$

$$c_3 = a_0 \cdot b_3 + a_1 \cdot b_2 + a_2 \cdot b_1 + a_3 \cdot b_0$$

$$c_4 = a_1 \cdot b_3 + a_2 \cdot b_2 + a_3 \cdot b_1$$

$$c_5 = a_2 \cdot b_3 + a_3 \cdot b_2$$

$$c_6 = a_3 \cdot b_3$$

✧ It is exactly the same as the schoolbook multiplication (but without modulo $\mathcal{R}_q$) $\rightarrow \mathcal{O}(n^2)$.

POSITIVE-WRAPPED CONVOLUTION (PWC)

✧ A special case of the convolution, when the result is $a \cdot b \pmod{x^n - 1}$:

$$c_j = \sum_{i=0}^j a_i \cdot b_{j-i} + \sum_{i=j+1}^{n-1} a_i \cdot b_{n+j-i}, \quad j = \{0, 1, 2, \dots, n-1\}$$

✧ So, for example for $n = 4$ and $a = [a_0, a_1, a_2, a_3]$ and $b = [b_0, b_1, b_2, b_3]$:

$$c_0 = a_0 \cdot b_0 + a_1 \cdot b_3 + a_2 \cdot b_2 + a_3 \cdot b_1$$

$$c_1 = a_0 \cdot b_1 + a_1 \cdot b_0 + a_2 \cdot b_3 + a_3 \cdot b_2$$

$$c_2 = a_0 \cdot b_2 + a_1 \cdot b_1 + a_2 \cdot b_0 + a_3 \cdot b_3$$

$$c_3 = a_0 \cdot b_3 + a_1 \cdot b_2 + a_2 \cdot b_1 + a_3 \cdot b_0$$

✧ The only thing left is to reduce the coefficients modulo q .

✧ But this is still $\mathcal{O}(n^2)$.

POSITIVE-WRAPPED CONVOLUTION (PWC)

- ✧ A special case of the convolution, when the result is $a \cdot b \pmod{x^n - 1}$:

$$c_j = \sum_{i=0}^j a_i \cdot b_{j-i} + \sum_{i=j+1}^{n-1} a_i \cdot b_{n+j-i}, \quad j = \{0, 1, 2, \dots, n-1\}$$

- ✧ So, for example for $n = 4$ and $a = [a_0, a_1, a_2, a_3]$ and $b = [b_0, b_1, b_2, b_3]$:

$$c_0 = a_0 \cdot b_0 + a_1 \cdot b_3 + a_2 \cdot b_2 + a_3 \cdot b_1$$

$$c_1 = a_0 \cdot b_1 + a_1 \cdot b_0 + a_2 \cdot b_3 + a_3 \cdot b_2$$

$$c_2 = a_0 \cdot b_2 + a_1 \cdot b_1 + a_2 \cdot b_0 + a_3 \cdot b_3$$

$$c_3 = a_0 \cdot b_3 + a_1 \cdot b_2 + a_2 \cdot b_1 + a_3 \cdot b_0$$

- ✧ The only thing left is to reduce the coefficients modulo q .
- ✧ But this is still $\mathcal{O}(n^2)$.

NEGATIVE-WRAPPED CONVOLUTION (NWC)

- ✧ A special case of the convolution, when the result is $a \cdot b \pmod{x^n + 1}$:

$$c_j = \sum_{i=0}^j a_i \cdot b_{j-i} - \sum_{i=j+1}^{n-1} a_i \cdot b_{n+j-i}, \quad j = \{0, 1, 2, \dots, n-1\}$$

- ✧ So, for example for $n = 4$ and $a = [a_0, a_1, a_2, a_3]$ and $b = [b_0, b_1, b_2, b_3]$:

$$c_0 = a_0 \cdot b_0 - a_1 \cdot b_3 - a_2 \cdot b_2 - a_3 \cdot b_1$$

$$c_1 = a_0 \cdot b_1 + a_1 \cdot b_0 - a_2 \cdot b_3 - a_3 \cdot b_2$$

$$c_2 = a_0 \cdot b_2 + a_1 \cdot b_1 + a_2 \cdot b_0 - a_3 \cdot b_3$$

$$c_3 = a_0 \cdot b_3 + a_1 \cdot b_2 + a_2 \cdot b_1 + a_3 \cdot b_0$$

- ✧ The only thing left is to reduce the coefficients modulo q .
- ✧ This is the equation we want in PQC. But it's still $\mathcal{O}(n^2)$.

NEGATIVE-WRAPPED CONVOLUTION (NWC)

- ✧ A special case of the convolution, when the result is $a \cdot b \pmod{x^n + 1}$:

$$c_j = \sum_{i=0}^j a_i \cdot b_{j-i} - \sum_{i=j+1}^{n-1} a_i \cdot b_{n+j-i}, \quad j = \{0, 1, 2, \dots, n-1\}$$

- ✧ So, for example for $n = 4$ and $a = [a_0, a_1, a_2, a_3]$ and $b = [b_0, b_1, b_2, b_3]$:

$$c_0 = a_0 \cdot b_0 - a_1 \cdot b_3 - a_2 \cdot b_2 - a_3 \cdot b_1$$

$$c_1 = a_0 \cdot b_1 + a_1 \cdot b_0 - a_2 \cdot b_3 - a_3 \cdot b_2$$

$$c_2 = a_0 \cdot b_2 + a_1 \cdot b_1 + a_2 \cdot b_0 - a_3 \cdot b_3$$

$$c_3 = a_0 \cdot b_3 + a_1 \cdot b_2 + a_2 \cdot b_1 + a_3 \cdot b_0$$

- ✧ The only thing left is to reduce the coefficients modulo q .
- ✧ This is the equation we want in PQC. But it's still $\mathcal{O}(n^2)$.

TRANSFORMATION

- ✧ Discrete Fourier Transform (DFT) can compute the convolution faster. But it works on complex numbers.
- ✧ **Number Theoretic Transform (NTT)** is the solution for our problem. It is the generalization of DFT for finite fields or rings.
- ✧ Cyclic NTT for PWC, and **Negacyclic NTT** for NWC.

Polynomial Multiplication in NTT

- ✧ Transforming to NTT domain: $\hat{a} = \text{NTT}(a)$ and $\hat{b} = \text{NTT}(b)$.
- ✧ Pointwise multiplication in NTT domain: $\hat{c} = \hat{a} \circ \hat{b}$.
- ✧ Inverse NTT to get the result in the original domain: $c = \text{iNTT}(\hat{c})$.
 $\Rightarrow c = a \cdot b \pmod{x^n + 1}$.

EXAMPLE OF POLYNOMIAL MULTIPLICATION IN NTT

✧ $a = (2, 8, 4, 1), b = (5, 6, 7, 8)$

✧ $\mathcal{R}_q = \mathbb{Z}_{11}[x]/(x^4 + 1)$

NTT

✧ $\hat{a} = \text{NTT}(a) = (9, 7, 10, 1)$

✧ $\hat{b} = \text{NTT}(b) = (3, 4, 2, 8)$

Pointwise Multiplication (only 4 multiplications)

✧ $\hat{c} = \hat{a} \circ \hat{b} = (9, 7, 10, 1) \circ (3, 4, 2, 8) = (27, 28, 20, 8) \equiv (5, 6, 9, 8) \pmod{11}$

Inverse NTT

✧ $c = \text{iNTT}(\hat{c}) = (6, 8, 2, 4) \pmod{11}$

EXAMPLE OF POLYNOMIAL MULTIPLICATION IN NTT

$$\diamond a = (2, 8, 4, 1), \quad b = (5, 6, 7, 8)$$

$$\diamond \mathcal{R}_q = \mathbb{Z}_{11}[x]/(x^4 + 1)$$

NTT

$$\diamond \hat{a} = \text{NTT}(a) = (9, 7, 10, 1)$$

$$\diamond \hat{b} = \text{NTT}(b) = (3, 4, 2, 8)$$

Pointwise Multiplication (only 4 multiplications)

$$\diamond \hat{c} = \hat{a} \circ \hat{b} = (9, 7, 10, 1) \circ (3, 4, 2, 8) = (27, 28, 20, 8) \equiv (5, 6, 9, 8) \pmod{11}$$

Inverse NTT

$$\diamond c = \text{iNTT}(\hat{c}) = (6, 8, 2, 4) \pmod{11}$$

EXAMPLE OF POLYNOMIAL MULTIPLICATION IN NTT

✧ $a = (2, 8, 4, 1), b = (5, 6, 7, 8)$

✧ $\mathcal{R}_q = \mathbb{Z}_{11}[x]/(x^4 + 1)$

NTT

✧ $\hat{a} = \text{NTT}(a) = (9, 7, 10, 1)$

✧ $\hat{b} = \text{NTT}(b) = (3, 4, 2, 8)$

Pointwise Multiplication (only n multiplications)

✧ $\hat{c} = \hat{a} \circ \hat{b} = (9, 7, 10, 1) \circ (3, 4, 2, 8) = (27, 28, 20, 8) \equiv (5, 6, 9, 8) \pmod{11}$

Inverse NTT

✧ $c = \text{iNTT}(\hat{c}) = (6, 8, 2, 4) \pmod{11}$

EXAMPLE OF POLYNOMIAL MULTIPLICATION IN NTT

✧ $a = (2, 8, 4, 1), b = (5, 6, 7, 8)$

✧ $\mathcal{R}_q = \mathbb{Z}_{11}[x]/(x^4 + 1)$

NTT

✧ $\hat{a} = \text{NTT}(a) = (9, 7, 10, 1)$

✧ $\hat{b} = \text{NTT}(b) = (3, 4, 2, 8)$

Pointwise Multiplication (only n multiplications)

✧ $\hat{c} = \hat{a} \circ \hat{b} = (9, 7, 10, 1) \circ (3, 4, 2, 8) = (27, 28, 20, 8) \equiv (5, 6, 9, 8) \pmod{11}$

Inverse NTT

✧ $c = \text{iNTT}(\hat{c}) = (6, 8, 2, 4) \pmod{11}$

**But how many
multiplications are needed for
NTT and iNTT?**

NTT DEFINITION: $2n$ -TH ROOT OF UNITY

✧ First, we need to define the $2n$ -th root of unity ζ :

2n-th root of unity

$$\triangleright \zeta^{2n} \equiv 1 \pmod{q}$$

$$\triangleright \zeta^n \equiv -1 \pmod{q}$$

✧ For example, for $n = 4$, $q = 97$, $\zeta = 33$ is a valid $2n$ -th root of unity.

Example of ζ

$$33^8 \equiv 1 \pmod{97}$$

$$33^4 \equiv 96 \pmod{97}$$

NTT DEFINITION

✧ $\hat{a} = \text{NTT}(a) = (\hat{a}_0, \hat{a}_1, \dots, \hat{a}_{n-1})$:

NTT Equation

$$\hat{a}_j = \sum_{i=0}^{n-1} \zeta^{2ij+i} \cdot a_i \pmod{q}.$$

Example of $\text{NTT}(2, 3, 1, 1)$

$$\hat{a}_0 = 2 \cdot 1 + 8 \cdot 33 + 4 \cdot 33^2 + 1 \cdot 33^3 \pmod{97}$$

$$\hat{a}_1 = 2 \cdot 1 + 8 \cdot 33^3 + 4 \cdot 33^6 + 1 \cdot 33^9 \pmod{97}$$

$$\hat{a}_2 = 2 \cdot 1 + 8 \cdot 33^5 + 4 \cdot 33^{10} + 1 \cdot 33^{15} \pmod{97}$$

$$\hat{a}_3 = 2 \cdot 1 + 8 \cdot 33^7 + 4 \cdot 33^{14} + 1 \cdot 33^{21} \pmod{97}$$

NTT DEFINITION

✧ $\hat{a} = \text{NTT}(a) = (\hat{a}_0, \hat{a}_1, \dots, \hat{a}_{n-1})$:

NTT Equation

$$\hat{a}_j = \sum_{i=0}^{n-1} \zeta^{2ij+i} \cdot a_i \pmod{q}.$$

Example of NTT((2, 8, 4, 1))

$$\begin{aligned}\hat{a}_0 &= 2 \cdot 1 + 8 \cdot 33 + 4 \cdot 33^2 + 1 \cdot 33^3 \pmod{97} \\ \hat{a}_1 &= 2 \cdot 1 + 8 \cdot 33^3 + 4 \cdot 33^6 + 1 \cdot 33^9 \pmod{97} \\ \hat{a}_2 &= 2 \cdot 1 + 8 \cdot 33^5 + 4 \cdot 33^{10} + 1 \cdot 33^{15} \pmod{97} \\ \hat{a}_3 &= 2 \cdot 1 + 8 \cdot 33^7 + 4 \cdot 33^{14} + 1 \cdot 33^{21} \pmod{97}\end{aligned}$$

iNTT DEFINITION

✧ $a = \text{iNTT}(\hat{a}) = (a_0, a_1, \dots, a_{n-1})$:

iNTT Equation

$$a_i = n^{-1} \sum_{j=0}^{n-1} \zeta^{-2ij-i} \cdot \hat{a}_j \pmod{q}.$$

Example of $\text{iNTT}(9, 7, 10, 1)$

$$33^{-1} = 50 \pmod{97}$$

$$a_0 = 50 \cdot (9 \cdot 1 + 7 \cdot 33^{-1} + 10 \cdot 33^{-2} + 1 \cdot 33^{-3}) \pmod{97}$$

$$a_1 = 50 \cdot (9 \cdot 1 + 7 \cdot 33^{-3} + 10 \cdot 33^{-6} + 1 \cdot 33^{-9}) \pmod{97}$$

$$a_2 = 50 \cdot (9 \cdot 1 + 7 \cdot 33^{-5} + 10 \cdot 33^{-10} + 1 \cdot 33^{-15}) \pmod{97}$$

$$a_3 = 50 \cdot (9 \cdot 1 + 7 \cdot 33^{-7} + 10 \cdot 33^{-14} + 1 \cdot 33^{-21}) \pmod{97}$$

iNTT DEFINITION

✧ $a = \text{iNTT}(\hat{a}) = (a_0, a_1, \dots, a_{n-1})$:

iNTT Equation

$$a_i = n^{-1} \sum_{j=0}^{n-1} \zeta^{-2ij-i} \cdot \hat{a}_j \pmod{q}.$$

Example of iNTT((9, 7, 10, 1))

$$33^{-1} = 50 \pmod{97}$$

$$a_0 = 50 \cdot (9 \cdot 1 + 7 \cdot 33^{-1} + 10 \cdot 33^{-2} + 1 \cdot 33^{-3}) \pmod{97}$$

$$a_1 = 50 \cdot (9 \cdot 1 + 7 \cdot 33^{-3} + 10 \cdot 33^{-6} + 1 \cdot 33^{-9}) \pmod{97}$$

$$a_2 = 50 \cdot (9 \cdot 1 + 7 \cdot 33^{-5} + 10 \cdot 33^{-10} + 1 \cdot 33^{-15}) \pmod{97}$$

$$a_3 = 50 \cdot (9 \cdot 1 + 7 \cdot 33^{-7} + 10 \cdot 33^{-14} + 1 \cdot 33^{-21}) \pmod{97}$$

The number of multiplications for NTT multiplication

- ✧ n^2 multiplications for NTT.
- ✧ n multiplications for PWM (Pointwise Multiplication).
- ✧ $n^2 + n$ multiplications for iNTT.
- ✧ **Total: $2n^2 + n$ multiplications ($\mathcal{O}(n^2)$)!!!**

- ✧ Obviously, this is not the way.
- ✧ We need an efficient way to compute NTT and iNTT faster.

Periodicity and Symmetry of ζ

✧ Periodicity of ζ : $\zeta^8 = 1 \pmod{q} \Rightarrow \zeta^{k+8} \equiv \zeta^k$.

▷ For example: $\zeta^9 = \zeta \pmod{q}$.

✧ Symmetry: $\zeta^4 = -1 \pmod{q} \Rightarrow \zeta^{k+4} = -\zeta^k \pmod{q}$.

▷ For example: $\zeta^6 = -\zeta^2 \pmod{q}$.

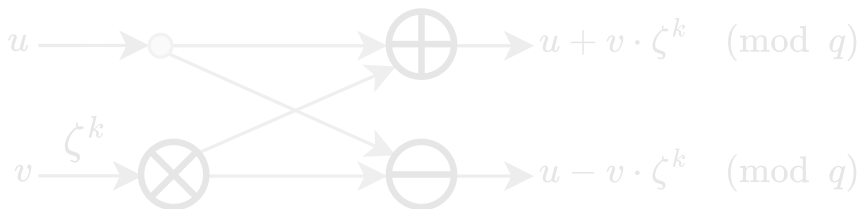
- ✧ We can rewrite the NTT equation using periodicity and symmetry properties of ζ .
- ✧ After rewriting and factoring out the common terms, a recursive pattern can be revealed.
- ✧ Based on the Fast Fourier Transform (FFT) algorithm, we can compute NTT faster:
 - ▷ **Cooley-Tukey (CT)** algorithm for NTT.
 - ▷ **Gentleman-Sande (GS)** algorithm for iNTT.

COOLEY-TUKEY (CT) BUTTERFLY ALGORITHM

CT Butterfly Pattern

$$u + v \cdot \zeta^k \pmod{q}$$

$$u - v \cdot \zeta^k \pmod{q}$$



CT Example

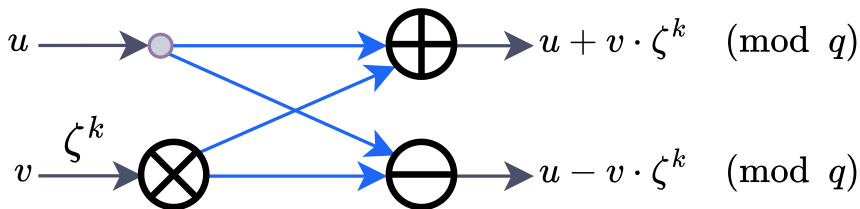
$$\text{CT}(u = 15, v = 27, \zeta = 33, k = 2) = (15 + 27 \cdot 33^2, 15 - 27 \cdot 33^2) = (27, 3) \pmod{97}$$

COOLEY-TUKEY (CT) BUTTERFLY ALGORITHM

CT Butterfly Pattern

$$u + v \cdot \zeta^k \pmod{q}$$

$$u - v \cdot \zeta^k \pmod{q}$$



CT Example

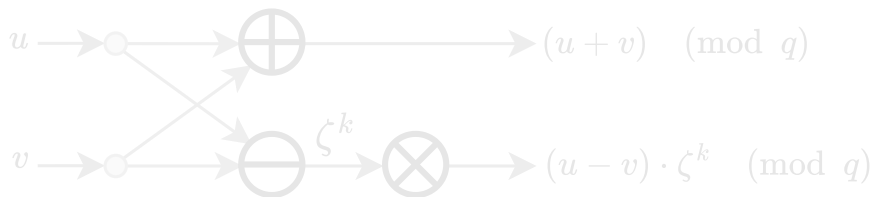
$$\text{CT}(u = 15, v = 27, \zeta = 33, k = 2) = (15 + 27 \cdot 33^2, 15 - 27 \cdot 33^2) = (27, 3) \pmod{97}$$

GENTLEMAN-SANDE (GS) BUTTERFLY ALGORITHM

GS Butterfly Pattern

$$u + v \pmod{q}$$

$$u - v \cdot \zeta^k \pmod{q}$$



GS Example

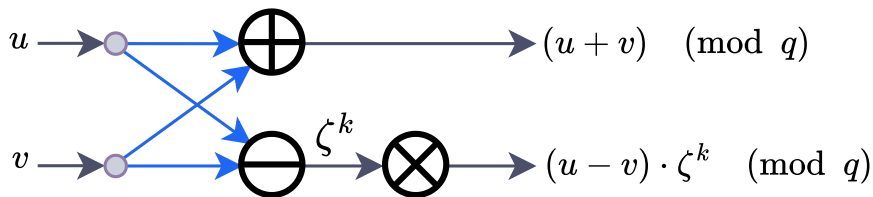
$$\text{GS}(u = 15, v = 27, \zeta = 33, k = 2) = (15 + 27, 15 - 27 \cdot 33^2) = (42, 27) \pmod{97}$$

GENTLEMAN-SANDE (GS) BUTTERFLY ALGORITHM

GS Butterfly Pattern

$$u + v \pmod{q}$$

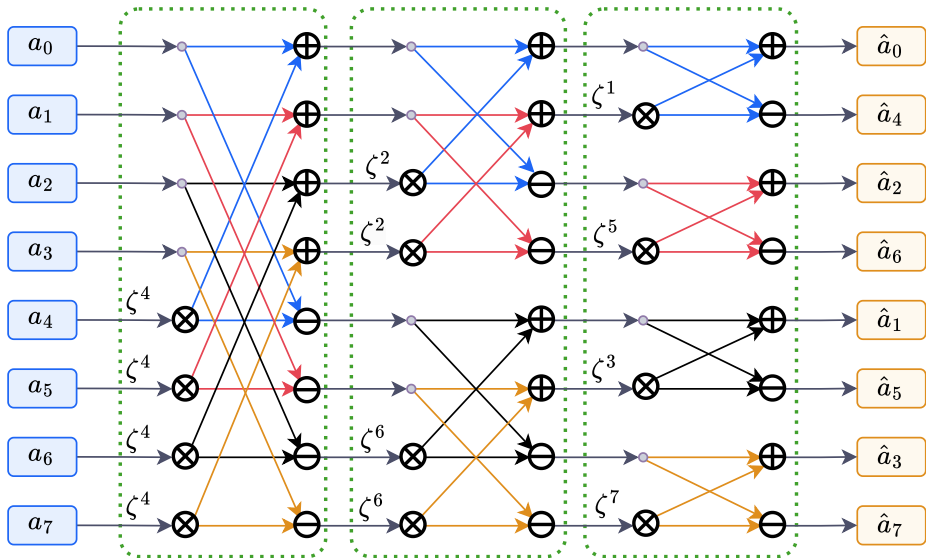
$$u - v \cdot \zeta^k \pmod{q}$$



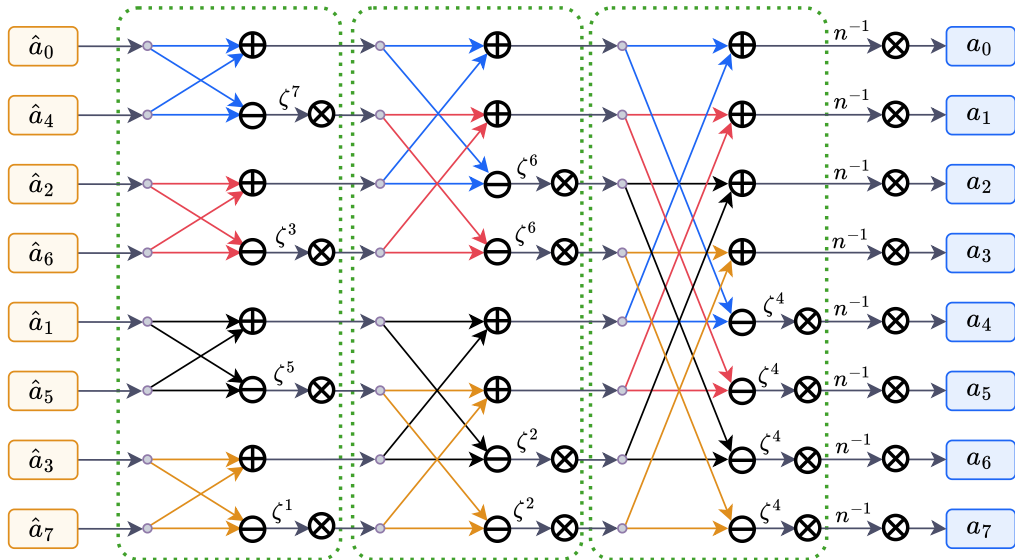
GS Example

$$\text{GS}(u = 15, v = 27, \zeta = 33, k = 2) = (15 + 27, 15 - 27 \cdot 33^2) = (42, 27) \pmod{97}$$

BUTTERFLY DIAGRAM OF 8-POINT NTT



BUTTERFLY DIAGRAM OF 8-POINT INTT

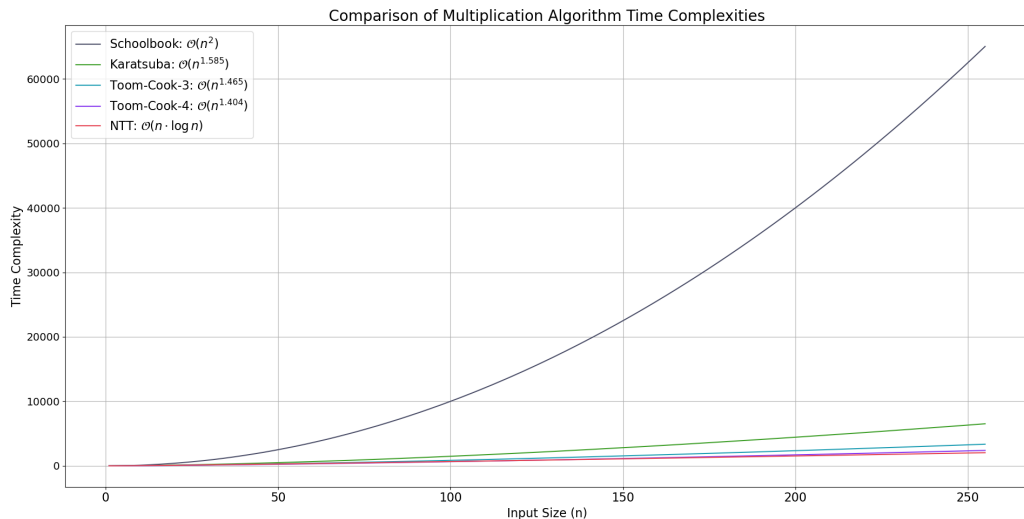


NUMBER OF MULTIPLICATIONS IN NTT

- ✧ NTT divides the problem into $\log_2 n$ stages, each stage requires $n/2$ multiplications.
 - ▷ $\frac{n}{2} \log_2 n$ multiplications for NTT.
 - ▷ n multiplications for PWM (Pointwise Multiplication).
 - ▷ $\frac{n}{2} \log_2 n$ multiplications for iNTT.
- ✧ **Total: $(n \log_2 n + n)$ multiplications $\rightarrow \mathcal{O}(n \log_2 n)$: Now it's fast!.**

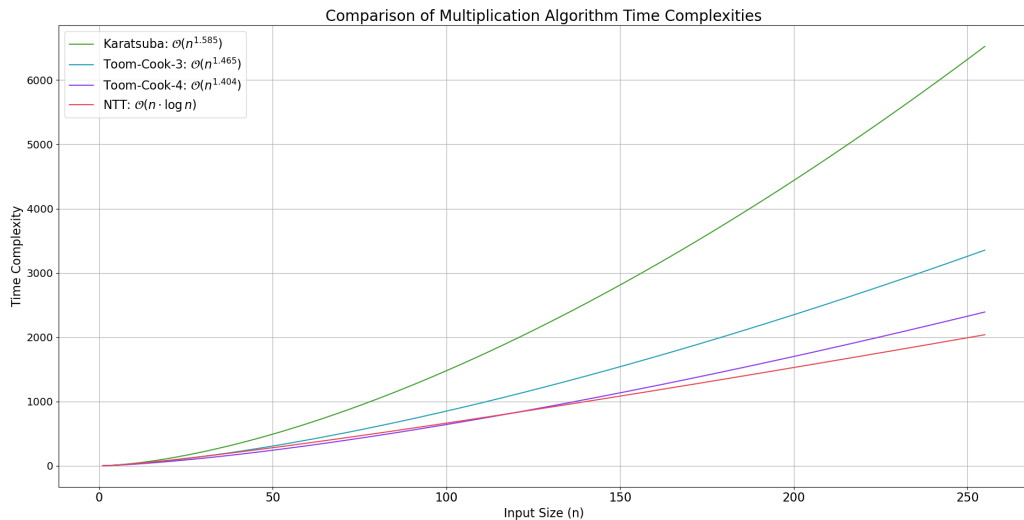
COMPARISON OF TIME COMPLEXITIES

✧ With Schoolbook.



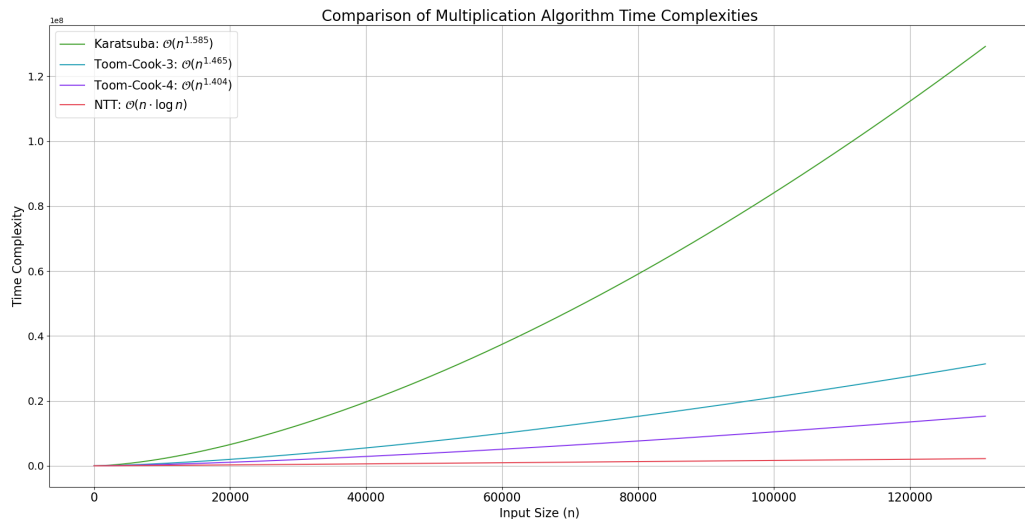
COMPARISON OF TIME COMPLEXITIES

✧ Without Schoolbook.



COMPARISON OF TIME COMPLEXITIES

✧ Without Schoolbook.



INPUT/OUTPUT ORDER IN NTT

- ✧ **BitRev_l(i)**: Reversing the bits of i in l -bit representation to efficiently compute butterfly operations:

$$l = 7, i = 0110101 = 53 \rightarrow \text{BitRev}_7(i) = 1010110 = 86$$

- ✧ CT butterfly:

- ▷ **Input**: Normal order coefficients (no).
- ▷ **Output**: *Bit-reversed order* coefficients (bo).

- ✧ GS butterfly:

- ▷ **Input**: *Bit-reversed order* coefficients (bo).
- ▷ **Output**: Normal order coefficients (no).

INPUT/OUTPUT ORDER IN NTT

- ✧ **BitRev_l(i)**: Reversing the bits of i in l -bit representation to efficiently compute butterfly operations:

$$l = 7, i = 0110101 = 53 \rightarrow \text{BitRev}_7(i) = 1010110 = 86$$

- ✧ CT butterfly:

- ▷ **Input**: Normal order coefficients (no).
- ▷ **Output**: *Bit-reversed order* coefficients (bo).

- ✧ GS butterfly:

- ▷ **Input**: *Bit-reversed order* coefficients (bo).
- ▷ **Output**: Normal order coefficients (no).

POWERS OF ζ IN NTT

- ✧ In NTT, for the butterfly unit number k , we have $\zeta^{\text{BitRev}_l(k)+1} \pmod{q}$, where l is the number of stages. So, for fixed n and q , powers of ζ are also fixed.
- ✧ For iNTT, we have $\zeta^{-(\text{BitRev}_l(k)+1)} \pmod{q}$. Also, we can access the powers of ζ for iNTT in a reverse order from the NTT.
- ✧ Powers of ζ are also called **twiddle factors**.
- ✧ If we have enough memory, we can precompute all the twiddle factors and store them in a ROM.

$$\zeta^{\text{BitRev}_8(0)} \mid \zeta^{\text{BitRev}_8(1)} \mid \zeta^{\text{BitRev}_8(2)} \mid \zeta^{\text{BitRev}_8(3)} \mid \dots \mid \zeta^{\text{BitRev}_8(254)} \mid \zeta^{\text{BitRev}_8(255)}$$

which is:

$$\zeta^0 \mid \zeta^{128} \mid \zeta^{64} \mid \zeta^{192} \mid \dots \mid \zeta^{127} \mid \zeta^{255}$$

- ✧ In Dilithium: $n = 256$, $q = 8380417$. Several $2n$ -th roots of unity exist $\rightarrow$ **1753** is the standard $2n$ -th root of unity in ML-DSA.
- ✧ $\log_2 256 = 8 \rightarrow$ 8 stages, each stage requires $n/2 = 128$ multiplications.
- ✧ Number of required multiplications to multiply two polynomials in ML-DSA:
 - ▷ Using schoolbook method: $n^2 = 256^2 = 2^{16} = 65536$ multiplications.
 - ▷ Using NTT: $n \log_2 n + n = 256 \log_2 256 + 256 = 2048 + 256 = 2304 \approx 2^{11}$ multiplications.
 $\rightarrow$ **28 times faster!**

✧ The goal is to compute $c = a \cdot b \pmod{\mathbb{Z}_{8380417}[x]/(x^{256} + 1)}$.

✧ Compute the **NTT** of a and b :

$$\hat{a} = (\hat{a}_0, \hat{a}_1, \dots, \hat{a}_{255})$$

$$\hat{b} = (\hat{b}_0, \hat{b}_1, \dots, \hat{b}_{255})$$

✧ Compute the **pointwise multiplication (PWM)** in the NTT domain:

$$\hat{c}_i = \hat{a}_i \cdot \hat{b}_i \pmod{q}$$

and we get:

$$\hat{c} = (\hat{c}_0, \hat{c}_1, \dots, \hat{c}_{255})$$

✧ Compute the **inverse NTT** of $\hat{c}$:

$$c = \text{iNTT}(\hat{c})$$

Algorithm 41 $\text{NTT}(w)$

Computes the NTT.

Input: Polynomial $w(X) = \sum_{j=0}^{255} w_j X^j \in R_q$.

Output: $\hat{w} = (\hat{w}[0], \dots, \hat{w}[255]) \in T_q$.

```
1: for  $j$  from 0 to 255 do
2:    $\hat{w}[j] \leftarrow w_j$ 
3: end for
4:  $m \leftarrow 0$ 
5:  $len \leftarrow 128$ 
6: while  $len \geq 1$  do
7:    $start \leftarrow 0$ 
8:   while  $start < 256$  do
9:      $m \leftarrow m + 1$ 
10:     $z \leftarrow \text{zetas}[m]$ 
11:    for  $j$  from  $start$  to  $start + len - 1$  do
12:       $t \leftarrow (z \cdot \hat{w}[j + len]) \bmod q$ 
13:       $\hat{w}[j + len] \leftarrow (\hat{w}[j] - t) \bmod q$ 
14:       $\hat{w}[j] \leftarrow (\hat{w}[j] + t) \bmod q$ 
15:    end for
16:     $start \leftarrow start + 2 \cdot len$ 
17:  end while
18:   $len \leftarrow \lfloor len/2 \rfloor$ 
19: end while
20: return  $\hat{w}$ 
```

$$\triangleright z \leftarrow \zeta^{\text{BitRev}_8(m)} \bmod q$$

INVERSE NTT ALGORITHM IN ML-DSA

Algorithm 42 $\text{NTT}^{-1}(\hat{w})$

Computes the inverse of the NTT.

Input: $\hat{w} = (\hat{w}[0], \dots, \hat{w}[255]) \in T_q$.

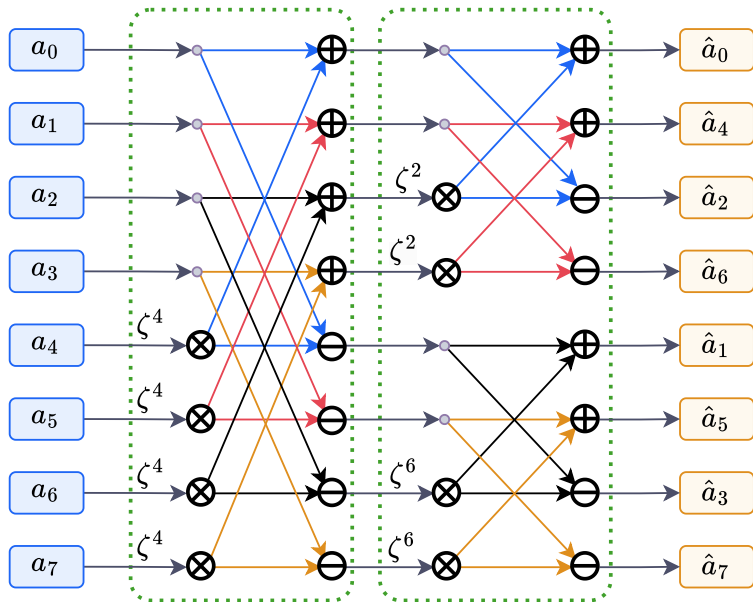
Output: Polynomial $w(X) = \sum_{j=0}^{255} w_j X^j \in R_q$.

```
1: for  $j$  from 0 to 255 do
2:    $w_j \leftarrow \hat{w}[j]$ 
3: end for
4:  $m \leftarrow 256$ 
5:  $len \leftarrow 1$ 
6: while  $len < 256$  do
7:    $start \leftarrow 0$ 
8:   while  $start < 256$  do
9:      $m \leftarrow m - 1$ 
10:     $z \leftarrow -\zeta^{\text{BitRev}_q(m)}$   $\text{mod } q$ 
11:    for  $j$  from  $start$  to  $start + len - 1$  do
12:       $t \leftarrow w_j$ 
13:       $w_j \leftarrow (t + w_{j+len}) \text{ mod } q$ 
14:       $w_{j+len} \leftarrow (t - w_{j+len}) \text{ mod } q$ 
15:       $w_{j+len} \leftarrow (z \cdot w_{j+len}) \text{ mod } q$ 
16:    end for
17:     $start \leftarrow start + 2 \cdot len$ 
18:  end while
19:   $len \leftarrow 2 \cdot len$ 
20: end while
21:  $f \leftarrow 8347681$ 
22: for  $j$  from 0 to 255 do
23:    $w_j \leftarrow (f \cdot w_j) \text{ mod } q$ 
24: end for
25: return  $w$ 
```

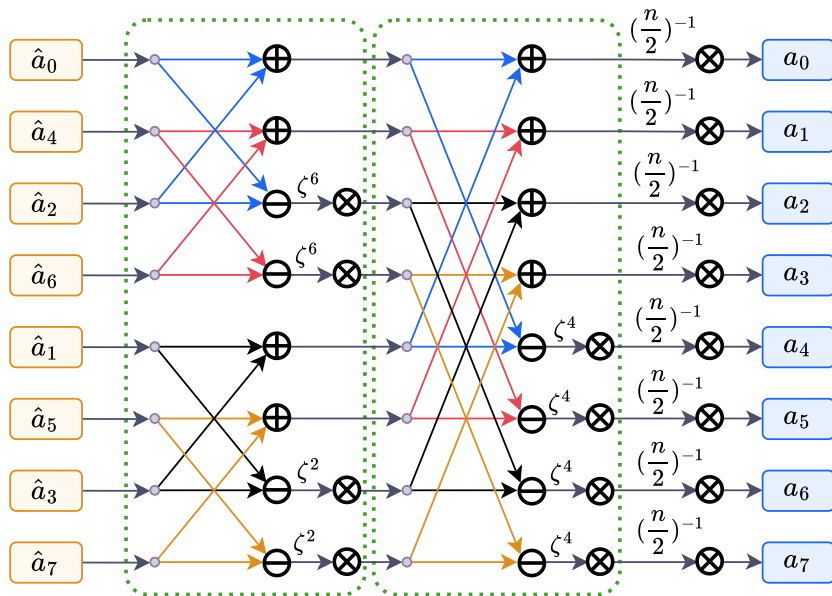
- ✧ For $n = 256$, $q = 3329$, $2n$ -th root of unity **does not exist**, but n -th root of unity exists.
→ We need some modifications to the NTT algorithm for Kyber.
- ✧ We will use ζ as the n -th root of unity → **17** is the 256^{th} root of unity in Kyber.
- ✧ To use Negacyclic NTT, the **incomplete** NTT and iNTT algorithms are used:
 - ▷ **For NTT**: Dataflow is the same as the complete NTT, but the last stage is skipped.
 - ▷ **For iNTT**: Dataflow is the same as the complete iNTT, but the first stage is skipped.
 - Also, at the end of the iNTT, we need to multiply the result by $\left(\frac{n}{2}\right)^{-1} \pmod{q}$, instead of $n^{-1} \pmod{q}$.
- ✧ So the number of stages is 7 (instead of 8) for Kyber's NTT and iNTT.

- ✧ For $n = 256$, $q = 3329$, $2n$ -th root of unity **does not exist**, but n -th root of unity exists.
→ We need some modifications to the NTT algorithm for Kyber.
- ✧ We will use ζ as the n -th root of unity → **17** is the 256^{th} root of unity in Kyber.
- ✧ To use Negacyclic NTT, the **incomplete** NTT and iNTT algorithms are used:
 - ▷ **For NTT**: Dataflow is the same as the complete NTT, but the **last stage** is skipped.
 - ▷ **For iNTT**: Dataflow is the same as the complete iNTT, but the **first stage** is skipped.
 - Also, at the end of the iNTT, we need to multiply the result by $\left(\frac{n}{2}\right)^{-1} \pmod{q}$, instead of $n^{-1} \pmod{q}$.
- ✧ So the number of stages is 7 (instead of 8) for Kyber's NTT and iNTT.

BUTTERFLY DIAGRAM OF 8-POINT INCOMPLETE NTT



BUTTERFLY DIAGRAM OF 8-POINT INCOMPLETE INTT



✧ The goal is to compute $c = a \cdot b \pmod{\mathbb{Z}_{3329}[x]/(x^{256} + 1)}$.

✧ Compute the **incomplete NTT** of a and b (in 7 stages):

$$\hat{a} = (\hat{a}_0, \hat{a}_1, \dots, \hat{a}_{255})$$

$$\hat{b} = (\hat{b}_0, \hat{b}_1, \dots, \hat{b}_{255})$$

✧ Compute the **pointwise multiplication (PWM)** in the NTT domain:

$$\hat{c}_{2i} = \hat{a}_{2i} \cdot \hat{b}_{2i} + \hat{a}_{2i+1} \cdot \hat{b}_{2i+1} \cdot 17^{2\text{BitRev}_7(i)+1} \pmod{q}$$

$$\hat{c}_{2i+1} = \hat{a}_{2i} \cdot \hat{b}_{2i+1} + \hat{a}_{2i+1} \cdot \hat{b}_{2i} \pmod{q}$$

and we get:

$$\hat{c} = (\hat{c}_0, \hat{c}_1, \dots, \hat{c}_{255})$$

✧ Compute the **inverse incomplete NTT** of $\hat{c}$:

$$c = \text{iNTT}(\hat{c})$$

MORE ONE PWM IN KYBER

- ✧ 5 multiplications are required to compute 2 elements of the product polynomial in the NTT domain:

Kyber's PWM Example

$$\hat{c}_0 = \hat{a}_0 \times \hat{b}_0 + \hat{a}_1 \times \hat{b}_1 \times 17^{2\text{BitRev}_7(0)+1} \pmod{q}$$

$$\hat{c}_1 = \hat{a}_0 \times \hat{b}_1 + \hat{a}_1 \times \hat{b}_0 \pmod{q}$$

- ✧ In addition to $\zeta^{\text{BitRev}_7(i)}$, we need to precompute $\zeta^{2\text{BitRev}_7(i)+1}$ for the PWM.
- ✧ So, the total number of multiplications to multiply two polynomials in Kyber is:
 $2 \times 7 \times 128 + 5 \times 128 = 2432$.

Algorithm 9 $\text{NTT}(f)$

Computes the NTT representation $\hat{f}$ of the given polynomial $f \in R_q$.

Input: array $f \in \mathbb{Z}_q^{256}$. ▷ the coefficients of the input polynomial

Output: array $\hat{f} \in \mathbb{Z}_q^{256}$. ▷ the coefficients of the NTT of the input polynomial

```
1:  $\hat{f} \leftarrow f$  ▷ will compute in place on a copy of input array
2:  $i \leftarrow 1$ 
3: for ( $len \leftarrow 128$ ;  $len \geq 2$ ;  $len \leftarrow len/2$ )
4:   for ( $start \leftarrow 0$ ;  $start < 256$ ;  $start \leftarrow start + 2 \cdot len$ )
5:      $zeta \leftarrow \zeta^{\text{BitRev}_7(i)} \bmod q$ 
6:      $i \leftarrow i + 1$ 
7:     for ( $j \leftarrow start$ ;  $j < start + len$ ;  $j++$ )
8:        $t \leftarrow zeta \cdot f[j + len]$  ▷ steps 8-10 done modulo  $q$ 
9:        $\hat{f}[j + len] \leftarrow \hat{f}[j] - t$ 
10:       $\hat{f}[j] \leftarrow \hat{f}[j] + t$ 
11:   end for
12: end for
13: end for
14: return  $\hat{f}$ 
```

INVERSE NTT ALGORITHM IN ML-KEM

Algorithm 10 $\text{NTT}^{-1}(\hat{f})$

Computes the polynomial $f \in R_q$ that corresponds to the given NTT representation $\hat{f} \in T_q$.

Input: array $\hat{f} \in \mathbb{Z}_q^{256}$.

▷ the coefficients of input NTT representation

Output: array $f \in \mathbb{Z}_q^{256}$.

▷ the coefficients of the inverse NTT of the input

1: $f \leftarrow \hat{f}$

▷ will compute in place on a copy of input array

2: $i \leftarrow 127$

3: **for** ($len \leftarrow 2$; $len \leq 128$; $len \leftarrow 2 \cdot len$)

4: **for** ($start \leftarrow 0$; $start < 256$; $start \leftarrow start + 2 \cdot len$)

5: $zeta \leftarrow \zeta^{\text{BitRev}_7(i)} \bmod q$

6: $i \leftarrow i - 1$

7: **for** ($j \leftarrow start$; $j < start + len$; $j++$)

8: $t \leftarrow f[j]$

9: $f[j] \leftarrow t + f[j + len]$

▷ steps 9-10 done modulo q

10: $f[j + len] \leftarrow zeta \cdot (f[j + len] - t)$

11: **end for**

12: **end for**

13: **end for**

14: $f \leftarrow f \cdot 3303 \bmod q$

▷ multiply every entry by $3303 \equiv 128^{-1} \bmod q$

15: **return** f

WHERE TO USE NTT?

- ✧ Short answer: **Always** use NTT for polynomial multiplication: NIST Standard!

Rule of Thumb

- ✧ If a polynomial has some specific structure (like $\mathbf{s}_i$ or $\mathbf{e}_i$ that have to be small), generate it in the normal domain and then transform it to the NTT domain for faster multiplication.
 - ✧ If a polynomial has no specific structure (like $\mathbf{A}_{ij}$ that is totally random), generate it in the NTT domain in the first place.
- ✧ So, we generate $\mathbf{s}$ in the normal domain and then transform it to the NTT domain. But for $\mathbf{A}$, we generate a $k \times k$ matrix of random coefficients, assuming the coefficients are already in the NTT domain. Either way, we **have only to use NTT for the multiplication**.

ML-KEM



- ✧ **ML-KEM** (Module-Lattice-Based Key-Encapsulation Mechanism Standard) is the NIST's standard for the key exchange.
- ✧ It uses Kyber-PKE building blocks.
- ✧ The basic form of Kyber is a **CPA-IND** secure system (passive attacker, can request encryption), but it is not secure against **CCA2-IND** attacks (active attacker, can request decryption).
- ✧ **Fujisaki-Okamoto (FO)** transformation transforms any CPA-IND secure system to a CCA2-IND secure system → **ML-KEM**: Provides the highest security level.
 - ▷ To make sure that the ciphertext is from Bob, Alice encrypts m again (with her public key) using $K\text{-PKE.encrypt}$, and compares the result with the ciphertext from Bob. If there is any difference, Alice will generate a totally random bit string as the shared key; otherwise, the key generated after decrypting m is the valid key.

- ✧ **ML-KEM** (Module-Lattice-Based Key-Encapsulation Mechanism Standard) is the NIST's standard for the key exchange.
- ✧ It uses Kyber-PKE building blocks.
- ✧ The basic form of Kyber is a **CPA-IND** secure system (passive attacker, can request encryption), but it is not secure against **CCA2-IND** attacks (active attacker, can request decryption).
- ✧ **Fujisaki-Okamoto (FO)** transformation transforms any CPA-IND secure system to a CCA2-IND secure system → **ML-KEM**: Provides the highest security level.
 - ▷ To make sure that the ciphertext is from Bob, Alice encrypts m again (with her public key) using $K\text{-PKE.encrypt}$, and compares the result with the ciphertext from Bob. If there is any difference, Alice will generate a totally random bit string as the shared key; otherwise, the key generated after decrypting m is the valid key.

- ✧ **ML-KEM** (Module-Lattice-Based Key-Encapsulation Mechanism Standard) is the NIST's standard for the key exchange.
- ✧ It uses Kyber-PKE building blocks.
- ✧ The basic form of Kyber is a **CPA-IND** secure system (passive attacker, can request encryption), but it is not secure against **CCA2-IND** attacks (active attacker, can request decryption).
- ✧ **Fujisaki-Okamoto (FO)** transformation transforms any CPA-IND secure system to a CCA2-IND secure system → **ML-KEM**: Provides the highest security level.
 - ▷ To make sure that the ciphertext is from Bob, Alice encrypts m again (with her public key) using $K\text{-PKE.encrypt}$, and compares the result with the ciphertext from Bob. If there is any difference, Alice will generate a totally random bit string as the shared key; otherwise, the key generated after decrypting m is the valid key.

ML-KEM COMPONENTS

- ✧ **SampleNTT**: Generates a random polynomial in the NTT domain (like SampleA, but in the NTT domain).
- ✧ **SamplePolyCBD**: Samples a random polynomial with coefficients from a centered binomial distribution (CBD) to make a **small polynomial** $\rightarrow$ Symmetric about 0 $\rightarrow$ No bias in the generated noise.
- ✧ **PRF**: Pseudo-Random Function: Generates a random polynomial. It uses SHAKE256 to produce one 512η -bit output:

$$\text{PRF}_{\eta}(\sigma) := \text{SHAKE256}(\sigma, 512 \times \eta)$$

ML-KEM COMPONENTS

- ✧ **SampleNTT**: Generates a random polynomial in the NTT domain (like SampleA, but in the NTT domain).
- ✧ **SamplePolyCBD**: Samples a random polynomial with coefficients from a centered binomial distribution (CBD) to make a **small polynomial** $\rightarrow$ Symmetric about 0 $\rightarrow$ No bias in the generated noise.
- ✧ **PRF**: Pseudo-Random Function: Generates a random polynomial. It uses SHAKE256 to produce one 512η -bit output:

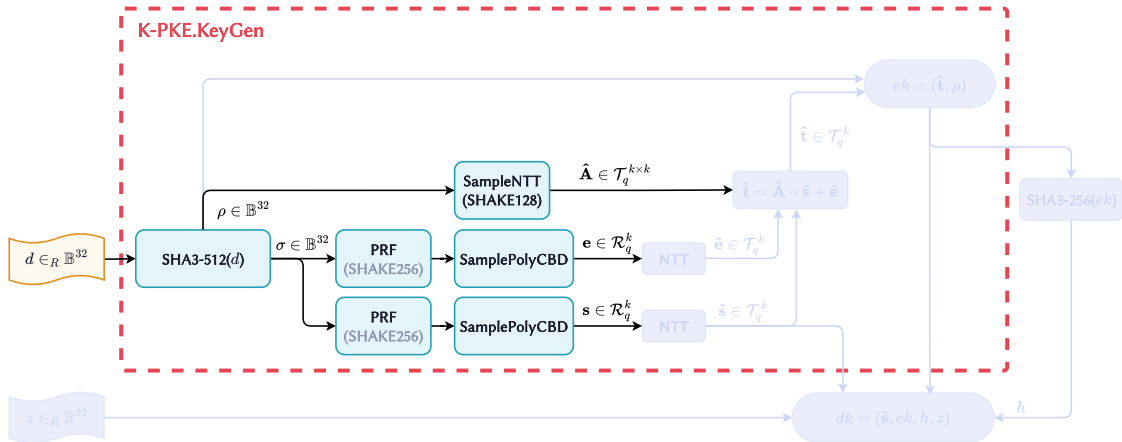
$$\text{PRF}_{\eta}(\sigma) := \text{SHAKE256}(\sigma, 512 \times \eta)$$

ML-KEM COMPONENTS

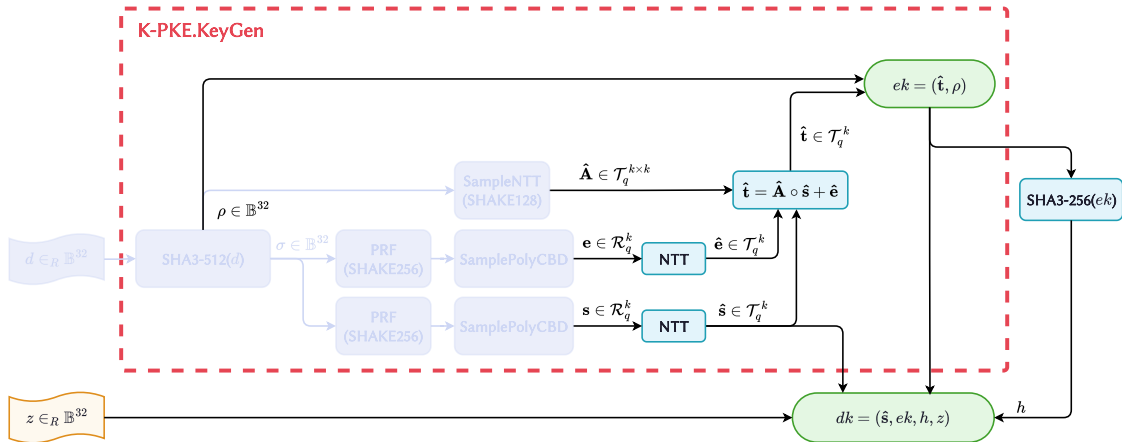
- ✧ **SampleNTT**: Generates a random polynomial in the NTT domain (like SampleA, but in the NTT domain).
- ✧ **SamplePolyCBD**: Samples a random polynomial with coefficients from a centered binomial distribution (CBD) to make a **small polynomial** $\rightarrow$ Symmetric about 0 $\rightarrow$ No bias in the generated noise.
- ✧ **PRF**: Pseudo-Random Function: Generates a random polynomial. It uses SHAKE256 to produce one 512η -bit output:

$$\text{PRF}_\eta(\sigma) := \text{SHAKE256}(\sigma, 512 \times \eta)$$

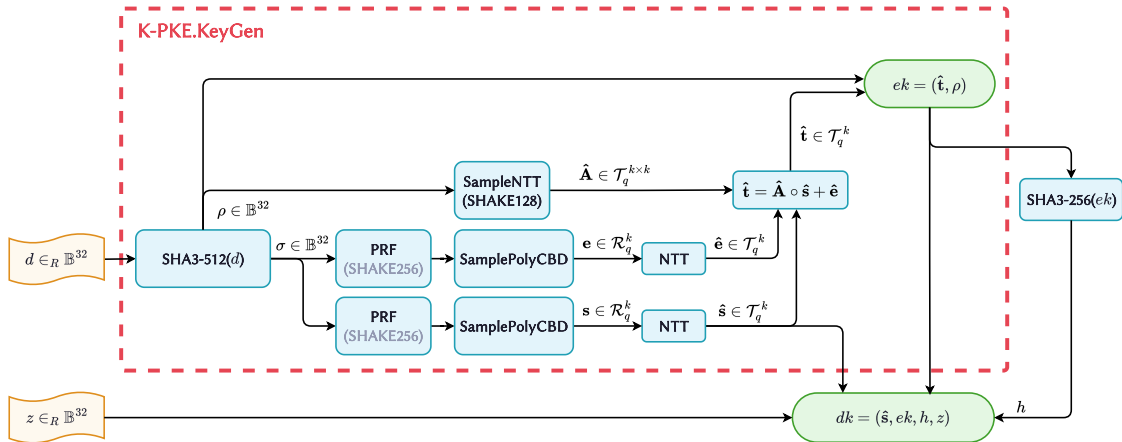
ML-KEM: KEY GENERATION



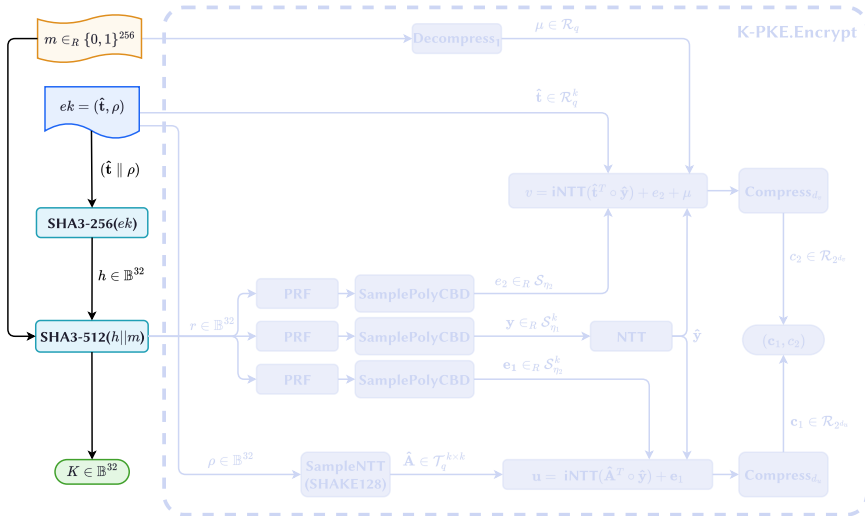
ML-KEM: KEY GENERATION



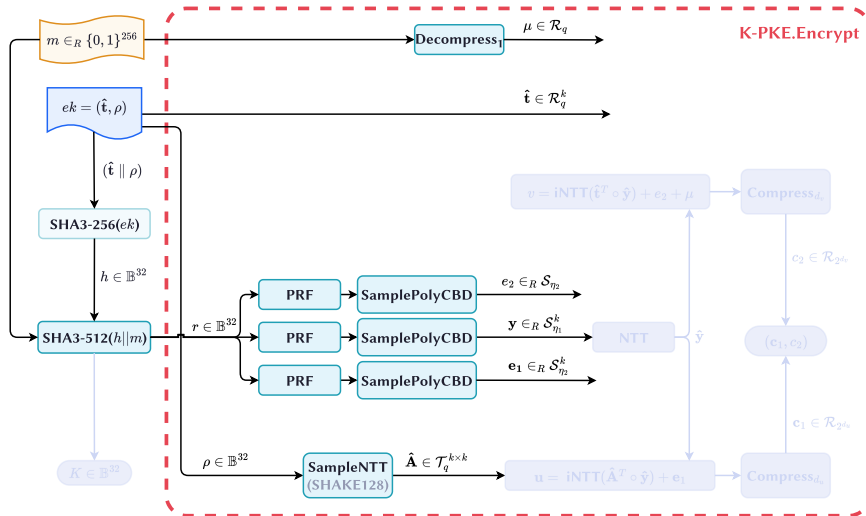
ML-KEM: KEY GENERATION



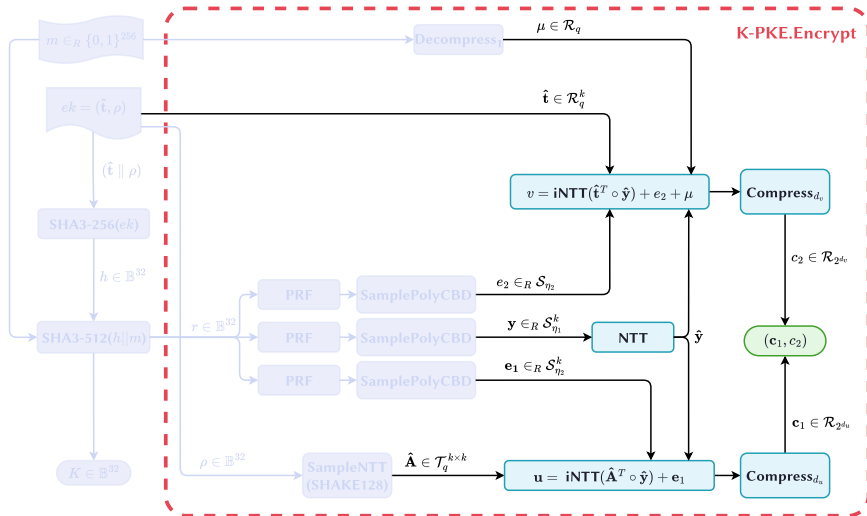
ML-KEM: ENCAPSULATION



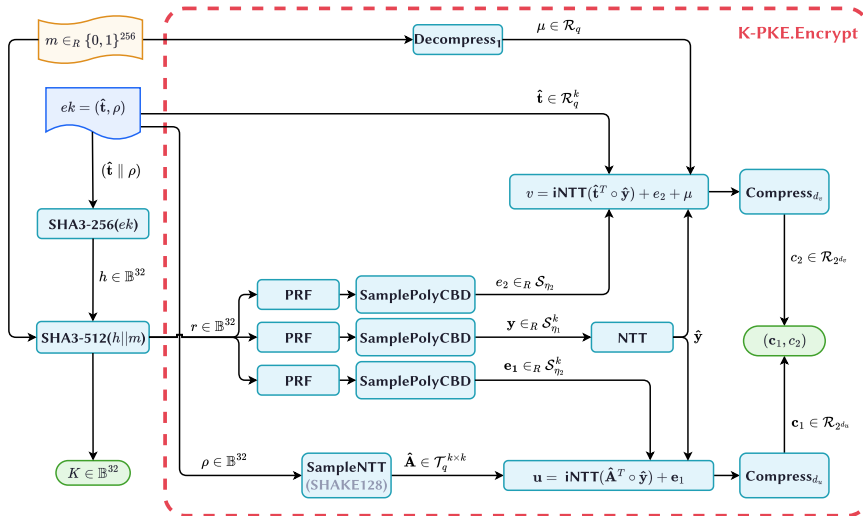
ML-KEM: ENCAPSULATION



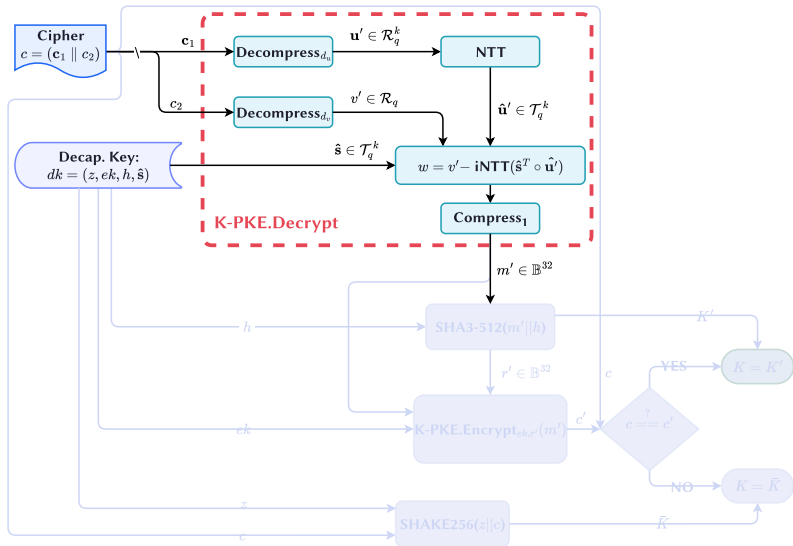
ML-KEM: ENCAPSULATION



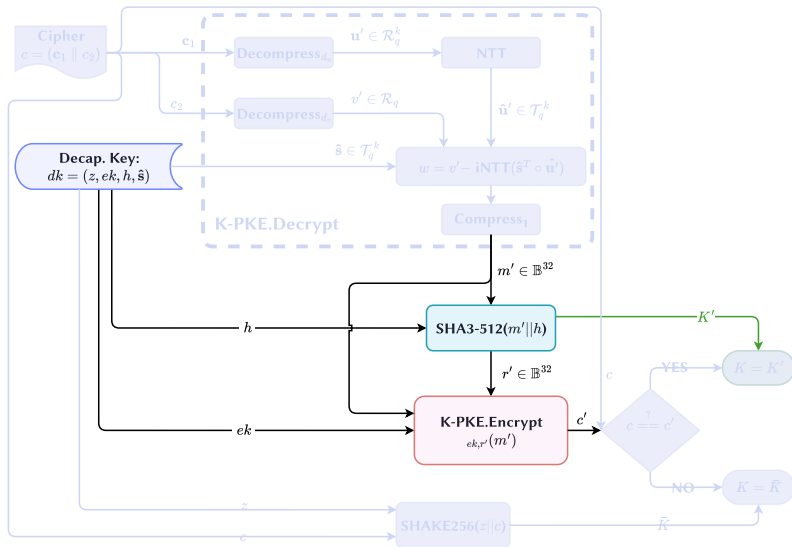
ML-KEM: ENCAPSULATION



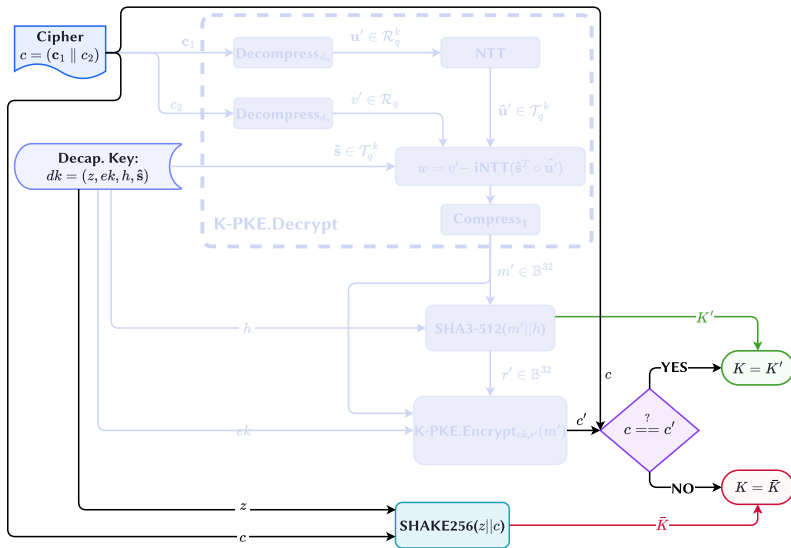
ML-KEM: DECAPSULATION



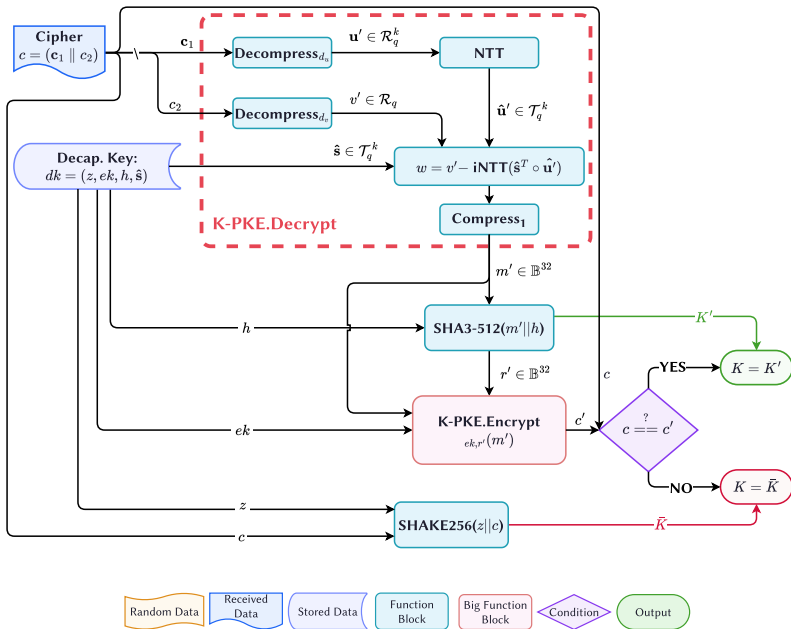
ML-KEM: DECAPSULATION



ML-KEM: DECAPSULATION



ML-KEM: DECAPSULATION



ML-KEM PARAMETER SETS

✧ Parameter sets named as ML-KEM- $(n \cdot k)$.

Approved Parameter Sets for ML-KEM

Schemes	n	q	k	η_1	η_2	d_u	d_v	ζ	Security Strength
ML-KEM-512	256	3329	2	3	2	10	4	17	AES-128
ML-KEM-768	256	3329	3	2	2	10	4	17	AES-192
ML-KEM-1024	256	3329	4	2	2	11	5	17	AES-256

SIZE COMPARISON OF DIFFERENT KEMs

Required Sizes (Bytes) for KEM Schemes

Schemes	Public Key	Private Key	Ciphertext	Shared Secret Key
ML-KEM-512	800	1632	768	32
ML-KEM-768	1184	2400	1088	32
ML-KEM-1024	1568	3168	1568	32
RSA-KEM (2048-bit)	256	256	256	32
Diffie-Hellman (2048-bit)	256	256	256	32
ECDH (Curve25519)	32	32	32	32

FAILURE RATE OF DECAPSULATION

- ✧ Decapsulation uses the $\text{Compression}_1(\text{Decompression}_1(m))$ to decapsulate the ciphertext.
- ✧ Sometimes everything is correct, but decapsulation fails.
- ✧ The failure rate is not zero, but very low and negligible.

Decapsulation Failure Rate for ML-KEM

Parameter Set	Decapsulation Failure Rate
ML-KEM-512	$2^{-138.8} \approx 10^{-41.8}$
ML-KEM-768	$2^{-164.8} \approx 10^{-49.6}$
ML-KEM-1024	$2^{-174.8} \approx 10^{-52.6}$

NOTES ON ML-DSA

FULL ML-DSA ALGORITHM

- ✧ Explanations on Dilithium in the last session were pretty comprehensive, and it covers approximately everything about ML-DSA. The only thing left is involving the NTT algorithm.
- ✧ As explained, for polynomials with a certain structure, we generate them in the normal domain, and then we use NTT. For $\mathbf{A}$, we generate a $k \times l$ matrix of random coefficients, assuming the coefficients are already in the NTT domain. Ultimately, we should use NTT for all the multiplications.

ML-DSA PARAMETER SETS

- ✧ $q = 2^{23} - 2^{13} + 1 = 8380417$ for all schemes.
- ✧ The NTT 512th root of unity $\zeta = 1753$ for all schemes.
- ✧ The number of dropped bits from $\mathbf{t}$, $d = 13$ for all schemes.
- ✧ Parameter sets named as ML-DSA- (kl) .

Approved Parameter Sets for ML-DSA

Schemes	λ	γ_1	γ_2	k	ℓ	η	τ	β	ω	Security Strength
ML-DSA-44	128	2^{17}	$\frac{q-1}{88}$	4	4	2	39	78	80	SHA-256
ML-DSA-65	192	2^{19}	$\frac{q-1}{32}$	6	5	4	49	196	55	AES-192
ML-DSA-87	256	2^{19}	$\frac{q-1}{32}$	8	7	2	60	120	75	AES-256

COMPARISON OF SIGNATURE SIZES

Required Sizes (Bytes) for DSA Schemes

Schemes	Signing Key	Verification Key	Signature Size
ML-DSA-44	2560	1312	2420
ML-DSA-65	4032	1952	3309
ML-DSA-87	4896	2592	4627
RSA-3072	384	384	384
ECDSA-P256	32	64	64
FN-DSA-512	13953	1793	1280
SLH-DSA-SHA2-192s	-	48	16224
XMSS-SHA2_20_256	2573	64	2820

EXPECTED NUMBER OF ITERATIONS FOR REJECTION SAMPLING

✧ $P_1 = \text{Prob}(\|z\|_\infty < \gamma_1 - \beta) \approx e^{-256\ell\beta/\gamma_1} \rightarrow \text{For ML-DSA-87: } 0.66.$

✧ $P_2 = \text{Prob}(\|r_0\|_\infty < \gamma_2 - \beta) \approx e^{-256k\beta/\gamma_2} \rightarrow \text{For ML-DSA-87: } 0.39.$

✧ $P_3 = \text{Prob}(\| -c \cdot t_0 \|_\infty < \gamma_2) \& (\text{HammingWeight}(h) \leq \omega) \approx 0.98$ (without any formal proof).

✧ Total expected number of iterations: $\frac{1}{P_1 P_2 P_3}.$

Schemes	Expected Number of Iterations
ML-DSA-44	4.25
ML-DSA-65	5.1
ML-DSA-87	3.85

Thank you!