

POST-QUANTUM CRYPTOGRAPHY

LECTURE 2: PQC CRYPTOSYSTEMS

Florida Atlantic University

Fall 2025

- 1 Standardized PQC Cryptosystems Overview
- 2 Notation Used in ML-Based Cryptosystems
 - Ring of Polynomials
 - MLWE Example
- 3 Key Exchange Mechanism: Kyber
 - KEM Overview
 - Kyber Components
 - How Kyber Works?
 - Kyber-PKE Algorithm
- 4 Digital Signature Algorithm: Dilithium
 - DSA Overview
 - Dilithium Components
 - Toy Dilithium
 - How Dilithium Guarantees the Authenticity of the Signature
 - Problems of Toy Dilithium and Required Optimizations
 - Dilithium Algorithm
 - Security of Dilithium

STANDARDIZED PQC CRYPTOSYSTEMS OVERVIEW

NIST STANDARDIZED PQC CRYPTOSYSTEMS

- ✧ We can build a key exchange cryptosystem based on D-MLWE that is resistant to cryptanalysis, even on quantum computers.
- ✧ We can also build a digital signature scheme based on D-MLWE and SelfTargetMSIS.
- ✧ NIST's competition winners:
 - ▷ Key Exchange: CRYSTALS-Kyber (FIPS 203) → Standardized as Module-Lattice-Based Key-Encapsulation Mechanism Standard (**ML-KEM**).
 - ▷ Digital Signature: CRYSTALS-Dilithium (FIPS 204) → Standardized as Module-Lattice-Based Digital Signature Standard (**ML-DSA**).
 - ▷ Other winners:
 - Digital Signature: FALCON (**FN-DSA**, FIPS 206), SPHINCS+ (**SLH-DSA**, FIPS 205)
 - Software and Firmware Update: LMS, XMSS (SP 800-208)
 - Key Exchange (Alternative): HQC (**HQC-KEM**, FIPS 207)

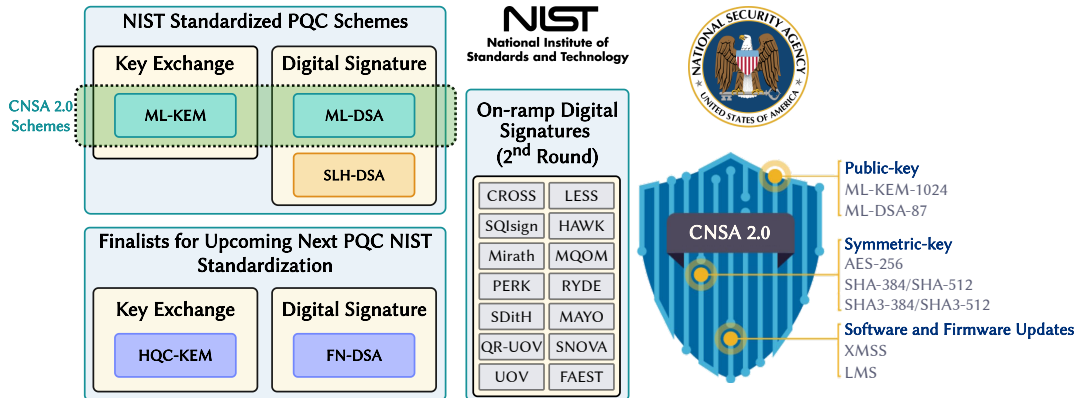
NIST STANDARDIZED PQC CRYPTOSYSTEMS

- ✧ We can build a key exchange cryptosystem based on D-MLWE that is resistant to cryptanalysis, even on quantum computers.
- ✧ We can also build a digital signature scheme based on D-MLWE and SelfTargetMSIS.
- ✧ NIST's competition winners:
 - ▷ **Key Exchange**: CRYSTALS-Kyber ([FIPS 203](#)) → Standardized as Module-Lattice-Based Key-Encapsulation Mechanism Standard (**ML-KEM**).
 - ▷ **Digital Signature**: CRYSTALS-Dilithium ([FIPS 204](#)) → Standardized as Module-Lattice-Based Digital Signature Standard (**ML-DSA**).
 - ▷ Other winners:
 - Digital Signature: FALCON (**FN-DSA**, FIPS 206), SPHINCS+ (**SLH-DSA**, FIPS 205)
 - Software and Firmware Update: LMS, XMSS (SP 800-208)
 - Key Exchange (Alternative): HQC (**HQC-KEM**, FIPS 207)

NIST STANDARDIZED PQC CRYPTOSYSTEMS

- ✧ We can build a key exchange cryptosystem based on D-MLWE that is resistant to cryptanalysis, even on quantum computers.
- ✧ We can also build a digital signature scheme based on D-MLWE and SelfTargetMSIS.
- ✧ NIST's competition winners:
 - ▷ **Key Exchange**: CRYSTALS-Kyber ([FIPS 203](#)) → Standardized as Module-Lattice-Based Key-Encapsulation Mechanism Standard (**ML-KEM**).
 - ▷ **Digital Signature**: CRYSTALS-Dilithium ([FIPS 204](#)) → Standardized as Module-Lattice-Based Digital Signature Standard (**ML-DSA**).
 - ▷ Other winners:
 - Digital Signature: FALCON (**FN-DSA**, [FIPS 206](#)), SPHINCS+ (**SLH-DSA**, [FIPS 205](#))
 - Software and Firmware Update: LMS, XMSS ([SP 800-208](#))
 - Key Exchange (Alternative): HQC (**HQC-KEM**, [FIPS 207](#))

NIST STANDARDIZED PQC CRYPTOSYSTEMS AND CNSA 2.0

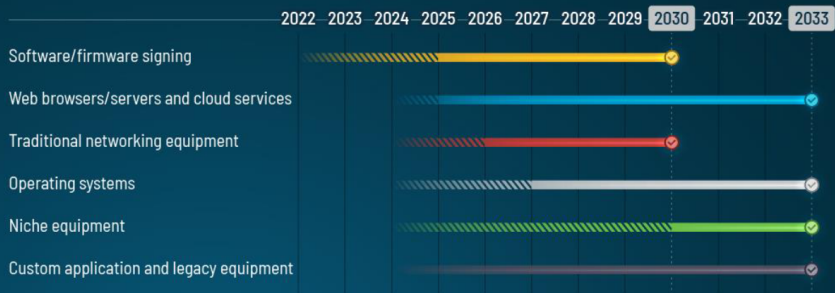


- FIPS-203:** Module-Lattice-Based Key-Encapsulation Mechanism Standard (**ML-KEM**, based on Crystals-Kyber)
- FIPS-204:** Module-Lattice-Based Key-Encapsulation Mechanism Standard (**ML-DSA**, based on Crystals-Dilithium)
- FIPS-205:** Stateless Hash-Based Digital Signature Standard (**SLH-DSA**, based on SPHINCS+)
- FIPS-206:** FFT over NTRU-Lattice-Based Digital Signature Algorithm (**FN-DSA**, based on FALCON)
- FIPS-207:** Hamming Quasi-Cyclic Key-Encapsulation Mechanism Standard (**HQC-KEM**, based on HQC)
- NIST SP 800-208:** Recommendation for Stateful Hash-Based Signature Schemes (**XMSS/LMS**)

CNSA 2.0 TIMELINE

✧ You can read more about CNSA 2.0 in the [CNSA 2.0 FAQ](#) document.

CNSA 2.0 Timeline

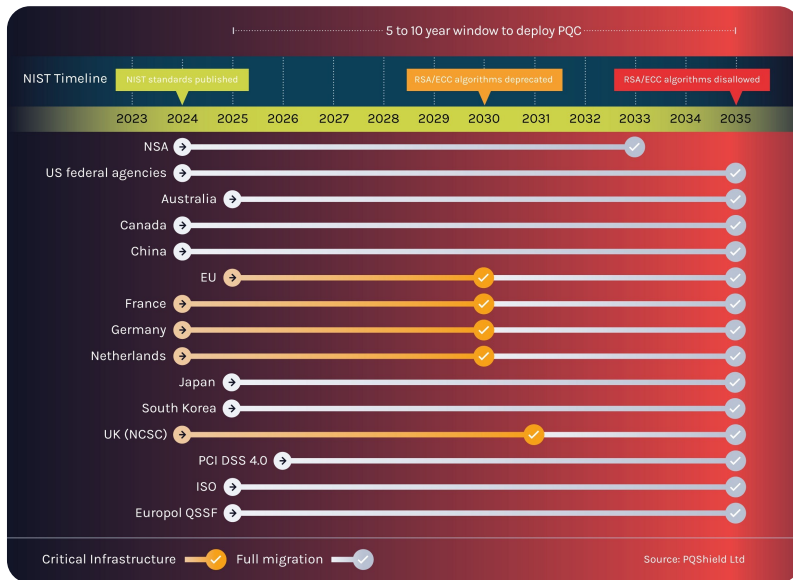


▨ CNSA 2.0 added as an option and tested

▬ CNSA 2.0 as the default and preferred

✓ Exclusively use CNSA 2.0 by this year

PQC Migration Around the World



NOTATION USED IN ML-BASED CRYPTOSYSTEMS

SYMMETRIC MOD

✧ $a \pmod{\pm q}$: Symmetric Mod $\rightarrow a \pmod{\pm q} \in [-\frac{q}{2}, \frac{q}{2}]$.

Symmetric Mod

if $a \leq \lfloor q/2 \rfloor$, then $a \pmod{\pm q} = a$,

if $a > \lfloor q/2 \rfloor$, then $a \pmod{\pm q} = a - q$.

Example

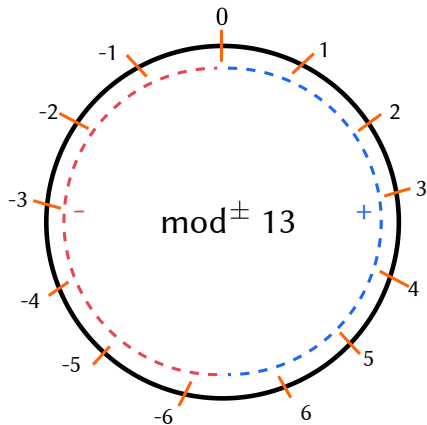
$$10 \equiv -3 \pmod{\pm 13}$$

$$4 \equiv 4 \pmod{\pm 13}$$

$$12 \equiv -1 \pmod{\pm 13}$$

$$21 \equiv 8 \equiv -5 \pmod{\pm 13}$$

$$32 \equiv 6 \equiv 6 \pmod{\pm 13}$$



SIZE OF POLYNOMIAL AND MATRIX

✧ $\|a\|_\infty$: Size of polynomial a : The maximum of coefficients of $a \pmod{\pm q}$.

Example ($q = 13$)

$$\|10x^2 + 4x + 1\|_\infty = \max(-3, 4, 1) = 4$$

✧ $\|\mathbf{a}\|_\infty$: Size of vector $\mathbf{a}$: The maximum of the sizes of the polynomials in $\mathbf{a}$. In other words, the maximum element in the vector after applying the symmetric mod.

Example ($q = 13$)

$$\|\mathbf{a}\|_\infty = \left\| \begin{bmatrix} 10x^2 + 4x + 1 \\ 3x^2 + 11x + 1 \\ 5x^2 + 1x + 1 \end{bmatrix} \right\|_\infty = \max(\max(-3, 4, 1), \max(3, -2, 1), \max(5, 1, 1)) = 5$$

SIZE OF POLYNOMIAL AND MATRIX

✧ $\|a\|_\infty$: Size of polynomial a : The maximum of coefficients of $a \pmod{\pm q}$.

Example ($q = 13$)

$$\|10x^2 + 4x + 1\|_\infty = \max(-3, 4, 1) = 4$$

✧ $\|\mathbf{a}\|_\infty$: Size of vector $\mathbf{a}$: The maximum of the sizes of the polynomials in $\mathbf{a}$. In other words, the maximum element in the vector after applying the symmetric mod.

Example ($q = 13$)

$$\|\mathbf{a}\|_\infty = \left\| \begin{bmatrix} 10x^2 + 4x + 1 \\ 3x^2 + 11x + 1 \\ 5x^2 + 1x + 1 \end{bmatrix} \right\|_\infty = \max(\max(-3, 4, 1), \max(3, -2, 1), \max(5, 1, 1)) = 5$$

RING OF POLYNOMIALS

✧ $\mathcal{R}_q^k$: Matrix of polynomials with k elements over the ring $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$.

✧ Example: a polynomial in $\mathcal{R}_{3329} = \mathbb{Z}_{3329}[x]/(x^{256} + 1)$:

$$123 + 1039x + 97x^2 + 2410x^3 + 987x^4 + 1446x^5 + \dots + 1963x^{254} + 396x^{255}$$

✧ We can also represent a polynomial as a vector of coefficients modulo q :

123	1039	97	2410	987	1446	...	1963	396
-----	------	----	------	-----	------	-----	------	-----

RING OF POLYNOMIALS

✧ Example: a vector of polynomials in $\mathcal{R}_{3329}^3 = \mathbb{Z}_{3329}[x]/(x^{256} + 1)$:

$$123 + 1039x + 97x^2 + 2410x^3 + 987x^4 + 1446x^5 + \dots + 1963x^{254} + 396x^{255}$$

$$2175 + 660x + 3273x^2 + 1931x^3 + 792x^4 + 9x^5 + \dots + 802x^{254} + 1611x^{255}$$

$$604 + 2449x + 706x^2 + 2926x^3 + 2999x^4 + 712x^5 + \dots + 154x^{254} + 1201x^{255}$$

✧ And store it as a vector of vectors of coefficients modulo q :

123	1039	97	2410	987	1446	...	1963	396
2175	660	3273	1931	792	9	...	802	1611
604	2449	706	2926	2999	712	...	154	1201

RING OF SMALL POLYNOMIALS

- ✧ $\mathcal{S}_\eta$: Ring of small polynomials. Like $\mathcal{R}_q$, but after applying symmetric mod q , the coefficients are in $[-\eta, \eta]$ ($\|\mathcal{S}_\eta\|_\infty \leq \eta$, and η is a very small number).
- ✧ A polynomial in $\mathcal{S}_2$ in ring $\mathcal{R}_{3329}$ with coefficients in $[-2, 2] \equiv \{3327, 3328, 0, 1, 2\} \pmod{\pm 3329}$:

$$1 + 0x + 2x^2 - 1x^3 + 1x^4 - 2x^5 + \dots + 0x^{254} + 2x^{255}$$

or:

1	0	2	3328	1	3327	...	0	2
---	---	---	------	---	------	-----	---	---

- ✧ $\tilde{\mathcal{S}}_\gamma$: Set of *relatively* small polynomials in $\mathcal{R}_q$ with coefficients after applying symmetric mod are in $(-\gamma, \gamma]$ (γ is a large number, but still small compared to q).

RING OF SMALL POLYNOMIALS

- ✧ $\mathcal{S}_\eta$: Ring of small polynomials. Like $\mathcal{R}_q$, but after applying symmetric mod q , the coefficients are in $[-\eta, \eta]$ ($\|\mathcal{S}_\eta\|_\infty \leq \eta$, and η is a very small number).
- ✧ A polynomial in $\mathcal{S}_2$ in ring $\mathcal{R}_{3329}$ with coefficients in $[-2, 2] \equiv \{3327, 3328, 0, 1, 2\} \pmod{\pm 3329}$:

$$1 + 0x + 2x^2 - 1x^3 + 1x^4 - 2x^5 + \dots + 0x^{254} + 2x^{255}$$

or:

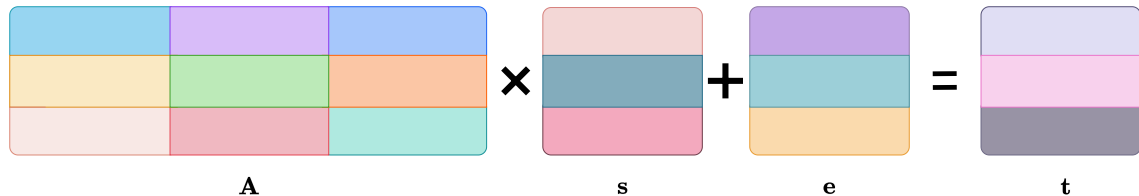
1	0	2	3328	1	3327	...	0	2
---	---	---	------	---	------	-----	---	---

- ✧ $\tilde{\mathcal{S}}_\gamma$: Set of *relatively* small polynomials in $\mathcal{R}_q$ with coefficients after applying symmetric mod are in $(-\gamma, \gamma]$ (γ is a large number, but still small compared to q).

MLWE EXAMPLE

✧ $\mathbf{A} \in \mathcal{R}_{3329}^{3 \times 3}$, $\mathbf{s} \in \mathcal{S}_{3329}^3$, $\mathbf{e} \in \mathcal{S}_{3329}^3$, $\mathbf{t} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$.

✧ Each colored box is a polynomial in $\mathbb{Z}_{3329}[x]/(x^{256} + 1)$:



✧ For example, t_0 can be calculated as:

$$t_0 = A_{00} \times s_0 + A_{01} \times s_1 + A_{02} \times s_2 + e_0$$

MLWE EXAMPLE

✧ $\mathbf{A} \in \mathcal{R}_{3329}^{3 \times 3}$, $\mathbf{s} \in \mathcal{S}_{3329}^3$, $\mathbf{e} \in \mathcal{S}_{3329}^3$, $\mathbf{t} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$.

✧ Each colored box is a polynomial in $\mathbb{Z}_{3329}[x]/(x^{256} + 1)$:

$\mathbf{A}$

$\mathbf{s}$

$\mathbf{e}$

$\mathbf{t}$

✧ For example, t_0 can be calculated as:

$$t_0 = A_{00} \times s_0 + A_{01} \times s_1 + A_{02} \times s_2 + e_0$$

$\text{mod } q$ $\text{mod } q$ $\text{mod } q$

MLWE EXAMPLE

✧ $\mathbf{A} \in \mathcal{R}_{3329}^{3 \times 3}, \mathbf{s} \in \mathcal{S}_{3329}^3, \mathbf{e} \in \mathcal{S}_{3329}^3, \mathbf{t} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}.$

✧ Each colored box is a polynomial in $\mathbb{Z}_{3329}[x]/(x^{256} + 1)$:

The diagram illustrates the equation $\mathbf{A} \cdot \mathbf{s} + \mathbf{e} = \mathbf{t}$. Matrix $\mathbf{A}$ is a 3×3 matrix with colored blocks. Vector $\mathbf{s}$ is a 3×1 vector. Vector $\mathbf{e}$ is a 3×1 vector. Vector $\mathbf{t}$ is a 3×1 vector. The result $\mathbf{t}$ is the sum of the product of $\mathbf{A}$ and $\mathbf{s}$, and $\mathbf{e}$.

✧ For example, t_0 can be calculated as:

The diagram illustrates the calculation of t_0 as the sum of products of $A_{00}, s_0, A_{01}, s_1, A_{02}, s_2$ and e_0 , all modulo q . The equation is:

$$t_0 = A_{00} \times s_0 + A_{01} \times s_1 + A_{02} \times s_2 + e_0$$

Each term $A_{ij} \times s_j$ is shown with a bracket above it labeled $\text{mod } \mathcal{R}_q$. Each s_j has an arrow pointing to it from below labeled $\text{mod } q$.

MLWE EXAMPLE

✧ $\mathbf{A} \in \mathcal{R}_{3329}^{3 \times 3}, \mathbf{s} \in \mathcal{S}_{3329}^3, \mathbf{e} \in \mathcal{S}_{3329}^3, \mathbf{t} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}.$

✧ Each colored box is a polynomial in $\mathbb{Z}_{3329}[x]/(x^{256} + 1)$:

$12 + 326x + \dots + 630x^{255}$	$1073 + 364x + \dots + 630x^{255}$	$625 + 2620x + \dots + 1302x^{255}$	$\times$	$2 + x + \dots + 0x^{255}$	$+$	$0 - 1x + \dots + 1x^{255}$	$=$	$1649 + 76x + \dots + 676x^{255}$
$1360 + 47x + \dots + 1237x^{255}$	$2562 + 637x + \dots + 2456x^{255}$	$1523 + 8x + \dots + 2303x^{255}$		$-1 + 0x + \dots + 2x^{255}$		$1 - 1x + \dots - 1x^{255}$		$310 + 1720x + \dots + 1600x^{255}$
$851 + 265x + \dots + 3146x^{255}$	$1009 + 1187x + \dots + 13x^{255}$	$195 + 2303x + \dots + 963x^{255}$		$-2 + 1x + \dots + 1x^{255}$		$1 - 2x + \dots + 2x^{255}$		$994 + 2492x + \dots + 521x^{255}$

$\mathbf{A} \qquad \qquad \mathbf{s} \qquad \qquad \mathbf{e} \qquad \qquad \mathbf{t}$

✧ For example, t_0 can be calculated as:

$$t_0 = \overbrace{A_{00} \times s_0}^{\text{mod } \mathcal{R}_q} + \overbrace{A_{01} \times s_1}^{\text{mod } \mathcal{R}_q} + \overbrace{A_{02} \times s_2}^{\text{mod } \mathcal{R}_q} + e_0$$

$\uparrow \qquad \qquad \uparrow \qquad \qquad \uparrow$
 $\text{mod } q \qquad \text{mod } q \qquad \text{mod } q$

OTHER NOTATIONS USED IN KYBER AND DILITHIUM

✧ $\lceil x \rceil$: Nearest integer number to x , ties broken upwards:

Example

$$\lceil 13.4 \rceil = 13, \lceil 13.6 \rceil = 14, \lceil -7.3 \rceil = -7, \lceil -7.8 \rceil = -8, \lceil 10.5 \rceil = 11, \lceil -10.5 \rceil = -10.$$

✧ $m \in_R \mathbb{B}^{32} \equiv m \xleftarrow{\$} \mathbb{B}^{32} \equiv m \xleftarrow{\$} \{0, 1\}^{256}$: m is 32 bytes (256 bits), randomly chosen ($\mathbb{B}$ is a byte).

Example

$$m = \overbrace{10000100100011111 \dots 1101101110001}^{256 \text{ bits}} \in \{0, 1\}^{256}$$

OTHER NOTATIONS USED IN KYBER AND DILITHIUM

- ✧ B_τ : Polynomials in $\mathcal{S}_1$, i.e., coefficients $(\text{mod}^\pm q) \in [-1, 0, 1]$, but exactly τ of them are ± 1 .

Example

$$B_6 = \overbrace{[-1, 0, 1, -1, 0, -1, 1, 0, 0, -1, 0, 0, \dots, 0]}^{256 \text{ bits}}$$

- ✧ $\beta = \tau \cdot \eta$, a parameter that is used to control the size of some of the polynomials.
- ✧ $\hat{c} = \hat{a} \circ \hat{b}$: Pointwise multiplication of $\hat{a}$ and $\hat{b}$:

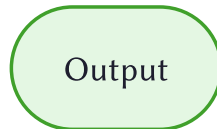
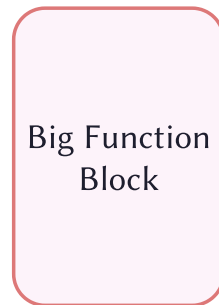
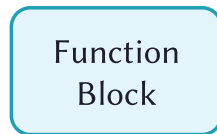
Example

$$\hat{a} = \{\hat{a}_0, \hat{a}_1, \dots, \hat{a}_{n-1}\} \in \mathcal{R}_q \text{ and } \hat{b} = \{\hat{b}_0, \hat{b}_1, \dots, \hat{b}_{n-1}\} \in \mathcal{R}_q$$

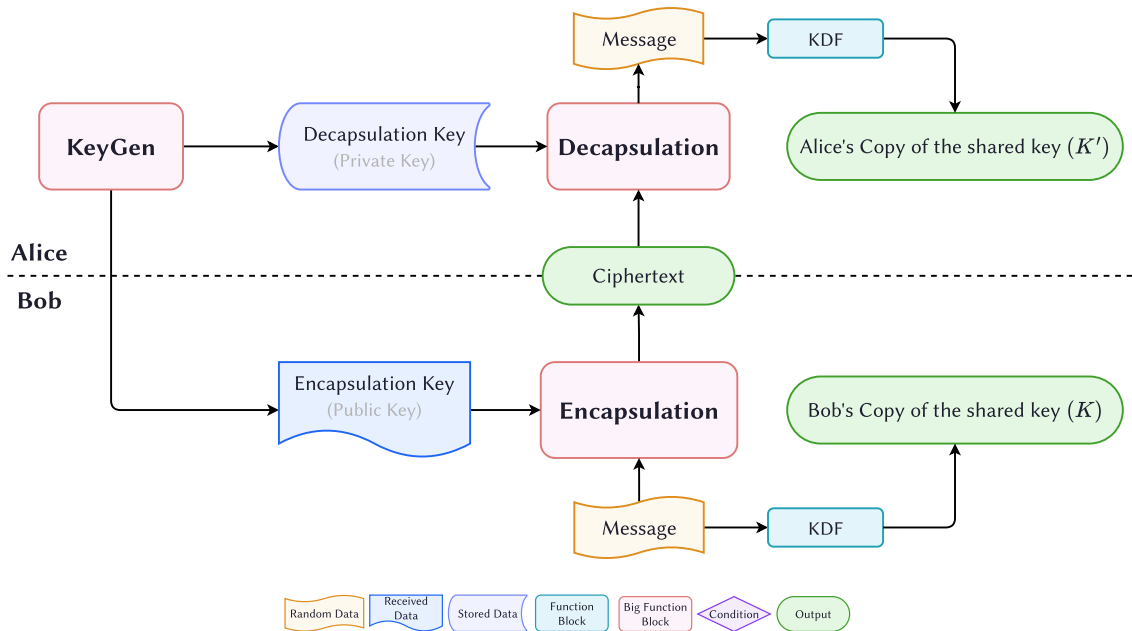
$$\Rightarrow \hat{c} = \hat{a} \circ \hat{b} = \{\hat{a}_0 \cdot \hat{b}_0, \hat{a}_1 \cdot \hat{b}_1, \dots, \hat{a}_{n-1} \cdot \hat{b}_{n-1}\} \in \mathcal{R}_q$$

KEY EXCHANGE MECHANISM: KYBER

HOW TO READ FLOWCHARTS



KEM OVERVIEW



KYBER: A POST-QUANTUM PUBLIC-KEY ENCRYPTION MECHANISM

- ✧ Kyber is based on the hardness of the Decisional-Module Learning With Errors (**D-MLWE**) problem → A quantum-resistant encryption mechanism.
- ✧ Kyber can encrypt any messages $m \in \mathcal{R}_q$, like RSA.
- ✧ **ML-KEM**: A complete secure key exchange scheme uses Kyber components.

KYBER: A POST-QUANTUM PUBLIC-KEY ENCRYPTION MECHANISM

- ✧ Kyber is based on the hardness of the Decisional-Module Learning With Errors (**D-MLWE**) problem → A quantum-resistant encryption mechanism.
- ✧ Kyber can encrypt any messages $m \in \mathcal{R}_q$, like RSA.
- ✧ **ML-KEM**: A complete secure key exchange scheme uses Kyber components.

KYBER: A POST-QUANTUM PUBLIC-KEY ENCRYPTION MECHANISM

- ✧ Kyber is based on the hardness of the Decisional-Module Learning With Errors (**D-MLWE**) problem → A quantum-resistant encryption mechanism.
- ✧ Kyber can encrypt any messages $m \in \mathcal{R}_q$, like RSA.
- ✧ **ML-KEM**: A complete secure key exchange scheme uses Kyber components.

RING OF POLYNOMIALS IN KYBER

Ring of Polynomials in Kyber

$$\mathcal{R}_{3329} = \mathbb{Z}_{3329}[x]/(x^{256} + 1)$$

$\Rightarrow$ Each element is a polynomial of degree 255, where the coefficients are integers modulo 3329.

$\Rightarrow$ A vector of length 256, where each element is a number in range $[0, 3329) \rightarrow 12 - \text{bit}$.

Example

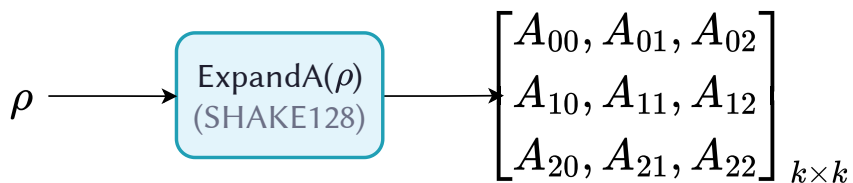
$$s_1 = 41 + 1246x + 310x^2 + \dots + 2341x^{255} \in \mathcal{R}_{3329}$$

or

$$s_1 = (\overbrace{41, 1246, 310, \dots, 2341}^{256}) \in \mathcal{R}_{3329}$$

KYBER COMPONENTS: EXPANDA

- ✧ **ExpandA:** Expands a random seed to a matrix of $k \times k$ polynomials **A**.
- ✧ Uses SHAKE128, a quantum-resistant hash function (XOF).



where $A_{ij} \in \mathcal{R}_q$ are polynomials, or vectors of n coefficients, each coefficient is an integer modulo q . So, the size of the matrix is $k \times k \times n \times \log_2(q)$ bits.

KYBER COMPONENTS: COMPRESS AND DECOMPRESS

✧ **Compress:** Compresses coefficients of a polynomial to a byte array: $a_i \in \mathbb{Z}_q \rightarrow \mathbb{Z}_{2^d}$:

$$\text{Compress}_d(x) = \lceil (\frac{2^d}{q}) \cdot x \rceil \pmod{2^d}$$

Example

$$\text{Compress}_{10}(2106) = \lceil (\frac{2^{10}}{3329}) \cdot 2106 \rceil \pmod{2^{10}} = 648$$

✧ **Decompress:** Decompresses a byte array to a polynomial: $a_i \in \mathbb{Z}_{2^d} \rightarrow \mathbb{Z}_q$:

$$\text{Decompress}_d(y) = \lceil (\frac{q}{2^d}) \cdot y \rceil$$

Example

$$\text{Decompress}_{10}(648) = \lceil (\frac{3329}{2^{10}}) \cdot 648 \rceil = 2106$$

KYBER COMPONENTS: COMPRESS AND DECOMPRESS

✧ **Compress:** Compresses coefficients of a polynomial to a byte array: $a_i \in \mathbb{Z}_q \rightarrow \mathbb{Z}_{2^d}$:

$$\text{Compress}_d(x) = \lceil (\frac{2^d}{q}) \cdot x \rceil \pmod{2^d}$$

Example

$$\text{Compress}_{10}(2106) = \lceil (\frac{2^{10}}{3329}) \cdot 2106 \rceil \pmod{2^{10}} = 648$$

✧ **Decompress:** Decompresses a byte array to a polynomial: $a_i \in \mathbb{Z}_{2^d} \rightarrow \mathbb{Z}_q$:

$$\text{Decompress}_d(y) = \lceil (\frac{q}{2^d}) \cdot y \rceil$$

Example

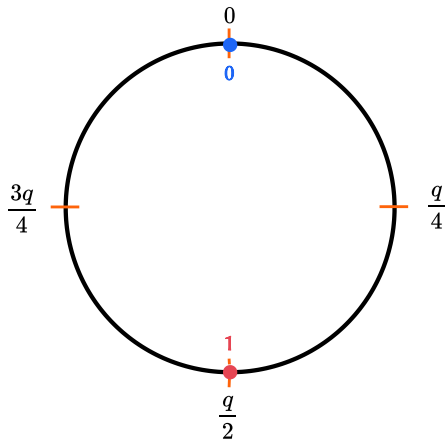
$$\text{Decompress}_{10}(648) = \lceil (\frac{3329}{2^{10}}) \cdot 648 \rceil = 2106$$

How Kyber Works?

DECOMPRESS₁(y)

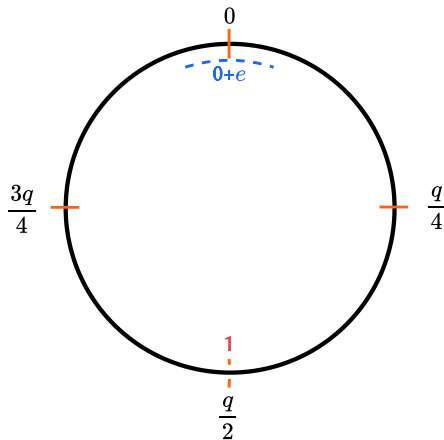
✧ $\text{Decompress}_1(y) = \lceil (\frac{q}{2}) \cdot y \rceil$: Enlarging a bit.

✧ $\text{Decompress}_1(0) = 0$ and $\text{Decompress}_1(1) = \frac{q}{2}$



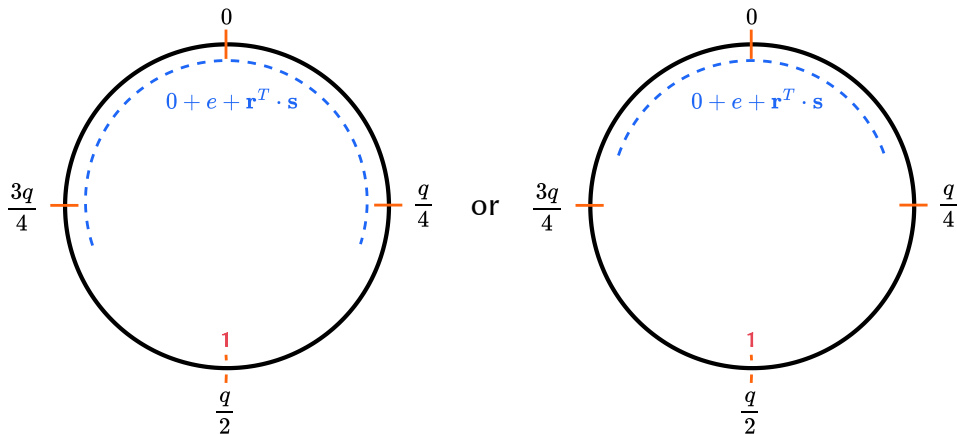
ADDING NOISES TO THE BIT

- ✧ Assume that the bit is 0.
- ✧ By adding a small noise to the decompressed bit, the value of the bit is changed to a random value, but still near zero (blue line is the range of $\text{Decompress}_1(y) + e$, when $y = 0$, and $e \in [-2, 2]$).



INVOLVING SECRET KEY TO THE NOISY BIT

- ✧ By adding a large number according to the secret key $\mathbf{s}$, the value of the bit is changed to a random value: maybe still near zero (the original bit), or far from it.
- ✧ $\mathbf{r} \in \mathcal{R}_q$, and $\mathbf{s} \in \mathcal{S}_\eta \Rightarrow \mathbf{r}^T \cdot \mathbf{s} \in [0, q)$, maybe small (right) or large (left).



COMPRESS₁(x)

✧ $\text{Compress}_1(x) = \lceil (\frac{2}{q}) \cdot x \rceil \pmod{2}$: A single bit.

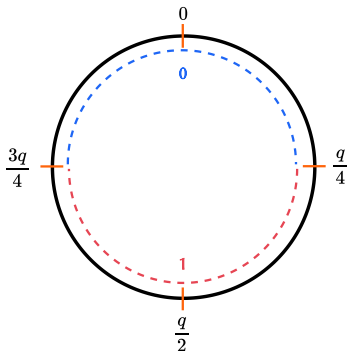
$$\text{✧ } \begin{cases} x < \frac{q}{4} \rightarrow x = \frac{q}{4} - \epsilon \Rightarrow \text{Compress}_1(x) = \lceil \frac{2}{q} \cdot (\frac{q}{4} - \epsilon) \rceil = \lceil \frac{1}{2} - \epsilon' \rceil = 0 \pmod{2} \\ x \geq \frac{q}{4} \rightarrow x = \frac{q}{4} + \epsilon \Rightarrow \text{Compress}_1(x) = \lceil \frac{2}{q} \cdot (\frac{q}{4} + \epsilon) \rceil = \lceil \frac{1}{2} + \epsilon' \rceil = 1 \pmod{2} \end{cases}$$

✧ And the same for $x = \frac{3q}{4} - \epsilon$ and $x = \frac{3q}{4} + \epsilon$.

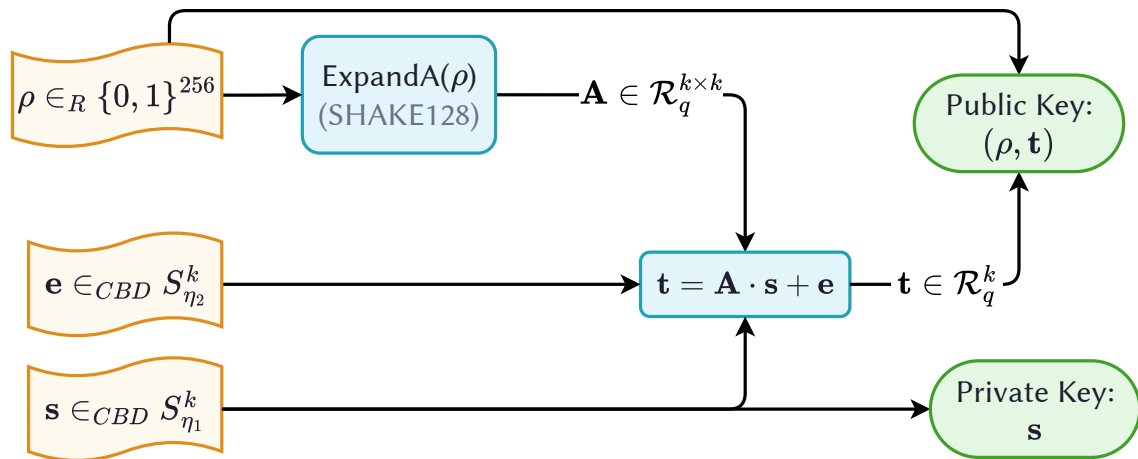
✧ So, it converts $x < \frac{q}{4}$ or $x \geq \frac{3q}{4}$ to 0, and $\frac{q}{4} \leq x < \frac{3q}{4}$ to 1.

✧ So, the only way to find the original bit is subtracting $\mathbf{r}^T \cdot \mathbf{s}$ to be in the correct range.

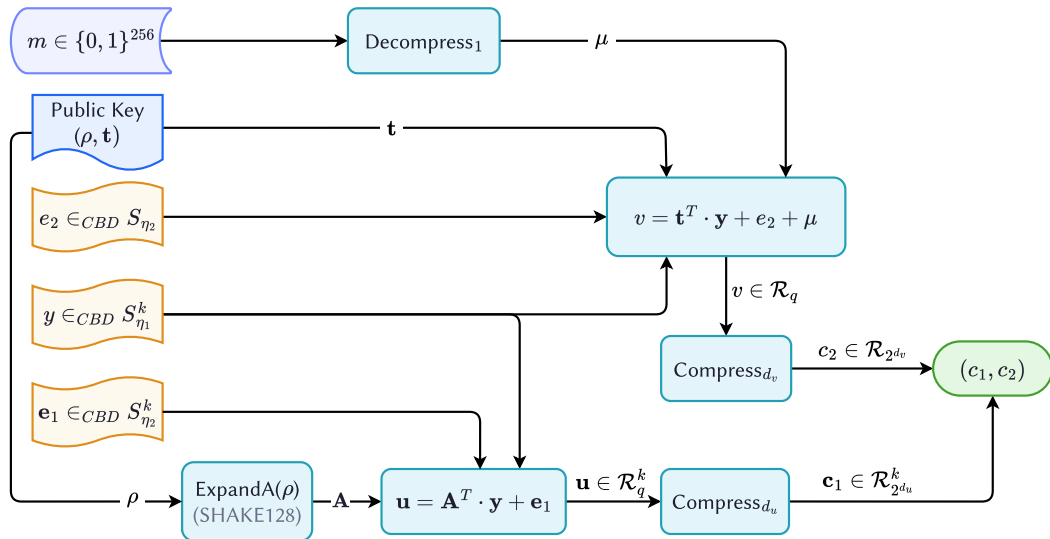
✧ Then, compress the result to get the original bit, which removes the noise e .



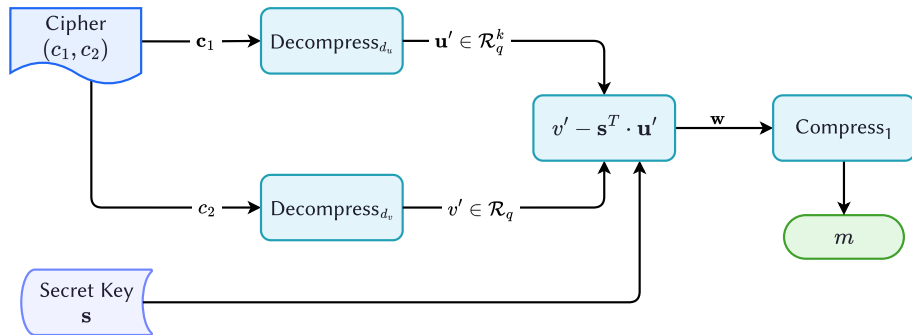
KYBER-PKE: KEY GENERATION (SIMPLIFIED)



KYBER-PKE: ENCRYPTION (SIMPLIFIED)



KYBER-PKE: DECRYPTION (SIMPLIFIED)



How KYBER-PKE WORKS

How Kyber-PKE Works

✧ Assume that compression and decompression is ideal: $\mathbf{u}' = \mathbf{u}$ and $v = v'$:

$$\mathbf{w} = \mathbf{v} - \mathbf{s}^T \mathbf{u} =$$

$$\mathbf{t}^T \mathbf{y} + e_2 + \mu - \mathbf{s}^T \mathbf{A}^T \mathbf{y} - \mathbf{s}^T \mathbf{e}_1 =$$

$$\cancel{\mathbf{s}^T \mathbf{A}^T \mathbf{y}} + \mathbf{e}^T \mathbf{y} + e_2 + \lceil \frac{q}{2} \cdot m \rceil - \cancel{\mathbf{s}^T \mathbf{A}^T \mathbf{y}} - \mathbf{s}^T \mathbf{e}_1$$

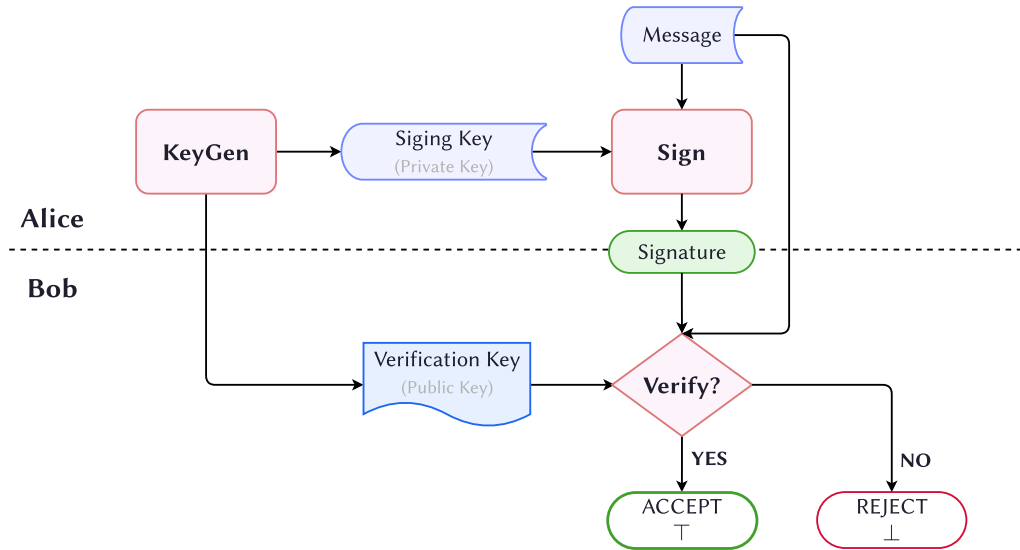
$$\Rightarrow \text{Compress}_1(\mathbf{w}) = \text{Compress}_1(\lceil \frac{q}{2} \cdot m \rceil - \mathbf{s}^T \mathbf{e}_1 + \mathbf{e}^T \mathbf{y} + e_2) \approx m$$

(Note that $\mathbf{e}$, $\mathbf{e}_1$, e_2 , $\mathbf{y}$ and $\mathbf{s}$ are small.)

$\Rightarrow$ The decryption is successful (*with very high probability!*).

DIGITAL SIGNATURE ALGORITHM: DILITHIUM

DSA OVERVIEW



DILITHIUM: A POST-QUANTUM DIGITAL SIGNATURE MECHANISM

- ✧ Dilithium is based on the hardness of the Module-Learning With Error (**MLWE**) and the modified nonstandard version of Module-Short Integer Solutions (**MSIS**) problem, called **SelfTargetMSIS**
→ A quantum-resistant digital signature mechanism.
- ✧ Dilithium is designed to achieve **Strong Existential Unforgeability under Chosen Message Attack (SUF-CMA)**:

SUF-CMA

Even if an attacker can get the signature of any message of their choice from an honest party, they are still unable to generate any additional valid signatures using the signer's public key, *even for messages that the signer has already signed*.

⇒ The attacker cannot generate a valid signature for a message that they have not signed before.

- ✧ **ML-DSA**: The NIST's standard for PQ digital signature: Dilithium with small changes.

DILITHIUM: A POST-QUANTUM DIGITAL SIGNATURE MECHANISM

- ✧ Dilithium is based on the hardness of the Module-Learning With Error (**MLWE**) and the modified nonstandard version of Module-Short Integer Solutions (**MSIS**) problem, called **SelfTargetMSIS**
→ A quantum-resistant digital signature mechanism.
- ✧ Dilithium is designed to achieve **Strong Existential Unforgeability under Chosen Message Attack** (SUF-CMA):

SUF-CMA

Even if an attacker can get the signature of any message of their choice from an honest party, they are still unable to generate any additional valid signatures using the signer's public key, *even for messages that the signer has already signed*.

⇒ The attacker cannot generate a valid signature for a message that they have not signed before.

- ✧ **ML-DSA**: The NIST's standard for PQ digital signature: Dilithium with small changes.

DILITHIUM: A POST-QUANTUM DIGITAL SIGNATURE MECHANISM

- ✧ Dilithium is based on the hardness of the Module-Learning With Error (**MLWE**) and the modified nonstandard version of Module-Short Integer Solutions (**MSIS**) problem, called **SelfTargetMSIS**
→ A quantum-resistant digital signature mechanism.
- ✧ Dilithium is designed to achieve **Strong Existential Unforgeability under Chosen Message Attack** (SUF-CMA):

SUF-CMA

Even if an attacker can get the signature of any message of their choice from an honest party, they are still unable to generate any additional valid signatures using the signer's public key, *even for messages that the signer has already signed*.

⇒ The attacker cannot generate a valid signature for a message that they have not signed before.

- ✧ **ML-DSA**: The NIST's standard for PQ digital signature: Dilithium with small changes.

Ring of Polynomials in Dilithium

$$\mathcal{R}_{8380417} = \mathbb{Z}_{8380417}[x]/(x^{256} + 1)$$

⇒ Each element is a polynomial of degree 255, where the coefficients are integers modulo 8380417.

⇒ A vector of length 256, where each element is a number in the range $[0, 8380417) \rightarrow 23 - \text{bit}$.

Example

$$s_1 = 36485 + 4153148x + 96734x^2 + \dots + 1636543x^{255} \in \mathcal{R}_{8380417}$$

or

$$s_1 = (36485, 4153148, 96734, \dots, 1636543) \in \mathcal{R}_{8380417}$$

✧ **MSIS:** Module-Short Integer Solution.

✧ Given:

▷ A norm bound γ (e.g. $\gamma = 100$),

▷ A matrix $\mathbf{A} \in_R \mathcal{R}_q^{k \times \ell}$,

▷ A nonzero module $\mathbf{y} \in \mathcal{S}_\gamma^\ell \longrightarrow \mathbf{y}$ is small: $0 < \|\mathbf{y}\|_\infty \leq \gamma$.

✧ Find a solution such that $\mathbf{A} \cdot \mathbf{y} \equiv 0 \pmod{q}$.

✧ Without the norm bound γ , this problem is trivial.

✧ Unlike MLWE, we don't have $\mathbf{e}$ here, but we have a norm bound γ , that makes this problem infeasible to solve.

- ✧ **SelfTargetMSIS** or **IM SIS** (Inhomogeneous Module-Short Integer Solution): Modified version of MSIS.
- ✧ Given:
 - ▷ A norm bound γ ,
 - ▷ A matrix $\mathbf{A} \in_R \mathcal{R}_q^{k \times \ell}$,
 - ▷ A nonzero module $\mathbf{y} \in \mathcal{S}_\gamma^\ell$,
 - ▷ A target module $\mathbf{t} \in \mathcal{R}_q^k$.
- ✧ Find a solution such that $\mathbf{A} \cdot \mathbf{y} \equiv \mathbf{t} \pmod{q}$.
- ✧ SelfTargetMSIS is at least as hard as SVP.
- ✧ A solver for MLWE can also solve SelfTargetMSIS, and a quantum solver for SelfTargetMSIS can also solve MLWE.

SCHNORR SIGNATURE SCHEME

- ✧ Schnorr Signature Scheme is a simple digital signature mechanism based on the hardness of the discrete logarithm problem (DLP).
- ✧ It uses a cyclic group of order n with a generator g .
- ✧ Like every digital signature mechanism, the Schnorr signature has three main components:
 - ▷ **Key Generation:** Alice generates a verification (public) key and a signing (private) key.
 - ▷ **Signature Generation:** Alice generates a signature for a message using the signing key.
 - ▷ **Signature Verification:** Bob verifies the signature using the verification key.

Algorithm 1: Schnorr Key Generation

Input: A group order n

Output: Alice's public key g^a and private key a

```
1  $a \xleftarrow{\$} [1, n - 1]$                                      ▷  $a$ : Private key
2  $pk \leftarrow g^a \pmod n$                                ▷  $pk$ : Public key
3 return  $pk, a$ 
```

SCHNORR SIGNATURE SCHEME: SIGNATURE GENERATION

✧ H : A secure hash function.

Algorithm 2: Schnorr Signature Generation

Input: A message $M \in \{0, 1\}^*$, Alice's private key a , and a group generator g .

Output: A signature $\sigma = (c, z)$.

1 $y \xleftarrow{\$} [1, n - 1]$

2 $w \leftarrow g^y$

3 $c \leftarrow H(M \parallel w)$

4 $z \leftarrow y + c \cdot a \pmod{n}$

5 $\sigma \leftarrow (c, z)$

6 **return** σ

▷ w : Commitment

▷ c : Challenge

▷ z : Response

▷ σ : Signature

SCHNORR SIGNATURE SCHEME: SIGNATURE VERIFICATION

Algorithm 3: Schnorr Signature Verification

Input: A message M , a signature $\sigma = (c, z)$, and Alice's public key g^a .

Output: Accept or reject the signature.

```
1  $w' \leftarrow g^z \cdot (g^a)^{-c}$ 
2 if  $c == H(M \parallel w')$  then
3   return 1                                ▷ Accept
4 return 0                                ▷ Reject
```

✧ It works, because:

$$w' = g^z \cdot (g^a)^{-c} = g^{y+c \cdot a} \cdot g^{-c \cdot a} = g^y = w \Rightarrow H(M \parallel w') = H(M \parallel w)$$

DILITHIUM COMPONENTS: LOWBITS AND HIGHBITS

- ✧ Considering an arbitrary number γ_2 , we can decompse each number r into two parts: low bits r_0 and high bits $r_1 \rightarrow$ by $r \pmod{\pm 2\gamma_2}$:

LowBits r_0 and HighBits r_1 Decomposition

$$r = r_1 \cdot 2\gamma_2 + r_0 \pmod{q}$$

$$-\gamma_2 \leq r_0 \leq \gamma_2, \quad 0 \leq r_1 < \frac{q-1}{2\gamma_2}$$

Example

$$r = 564, q = 817, \gamma_2 = 8 \rightarrow \frac{q-1}{2\gamma_2} = 51 \Rightarrow -8 \leq r_0 \leq 8, \quad 0 \leq r_1 < 51$$

$$r = 35 \cdot 2 \cdot 8 + 4 = 564 \Rightarrow r_1 = 35, r_0 = 4$$

DILITHIUM COMPONENTS: LOWBITS AND HIGHBITS

- ✧ Considering an arbitrary number γ_2 , we can decompose each number r into two parts: low bits r_0 and high bits $r_1 \rightarrow$ by $r \pmod{\pm 2\gamma_2}$:

LowBits r_0 and HighBits r_1 Decomposition

$$r = r_1 \cdot 2\gamma_2 + r_0 \pmod{q}$$

$$-\gamma_2 \leq r_0 \leq \gamma_2, \quad 0 \leq r_1 < \frac{q-1}{2\gamma_2}$$

Example

$$r = 564, q = 817, \gamma_2 = 8 \rightarrow \frac{q-1}{2\gamma_2} = 51 \Rightarrow -8 \leq r_0 \leq 8, \quad 0 \leq r_1 < 51$$

$$r = 35 \cdot 2 \cdot 8 + 4 = 564 \Rightarrow r_1 = 35, r_0 = 4$$

DILITHIUM COMPONENTS: LOWBITS AND HIGHBITS

- ✧ Considering an arbitrary number γ_2 , we can decompose each number r into two parts: low bits r_0 and high bits $r_1 \rightarrow$ by $r \pmod{\pm 2\gamma_2}$:

LowBits r_0 and HighBits r_1 Decomposition

$$r = r_1 \cdot 2\gamma_2 + r_0 \pmod{q}$$

$$-\gamma_2 \leq r_0 \leq \gamma_2, \quad 0 \leq r_1 < \frac{q-1}{2\gamma_2}$$

Example

$$r = 557, q = 817, \gamma_2 = 8 \rightarrow \frac{q-1}{2\gamma_2} = 51 \Rightarrow -8 \leq r_0 \leq 8, \quad 0 \leq r_1 < 51$$

$$r = 35 \cdot 2 \cdot 8 + -3 = 557 \Rightarrow r_1 = 35, r_0 = -3$$

DILITHIUM COMPONENTS: LOWBITS AND HIGHBITS

- ✧ Considering an arbitrary number γ_2 , we can decompose each number r into two parts: low bits r_0 and high bits $r_1 \rightarrow$ by $r \pmod{\pm 2\gamma_2}$:

LowBits r_0 and HighBits r_1 Decomposition

$$r = r_1 \cdot 2\gamma_2 + r_0 \pmod{q}$$

$$-\gamma_2 \leq r_0 \leq \gamma_2, \quad 0 \leq r_1 < \frac{q-1}{2\gamma_2}$$

Example

$$r = 816, q = 817, \gamma_2 = 8 \rightarrow \frac{q-1}{2\gamma_2} = 51 \Rightarrow -8 \leq r_0 \leq 8, \quad 0 \leq r_1 < 51$$

$$r = 51 \cdot 2 \cdot 8 + 0 = 816 \Rightarrow r_1 = 51, r_0 = 0$$

WRONG!

DILITHIUM COMPONENTS: LOWBITS AND HIGHBITS

- ✧ Considering an arbitrary number γ_2 , we can decompose each number r into two parts: low bits r_0 and high bits $r_1 \rightarrow$ by $r \pmod{\pm 2\gamma_2}$:

LowBits r_0 and HighBits r_1 Decomposition

$$r = r_1 \cdot 2\gamma_2 + r_0 \pmod{q}$$

$$-\gamma_2 \leq r_0 \leq \gamma_2, \quad 0 \leq r_1 < \frac{q-1}{2\gamma_2}$$

Example

$$r = 816, q = 817, \gamma_2 = 8 \rightarrow \frac{q-1}{2\gamma_2} = 51 \Rightarrow -8 \leq r_0 \leq 8, \quad 0 \leq r_1 < 51$$

$$r = 51 \cdot 2 \cdot 8 + 0 = 816 \Rightarrow r_1 = 51, r_0 = 0 \Rightarrow r_1 = 0, r_0 = -1$$

CORRECT!

DILITHIUM COMPONENTS: DECOMPOSITION ALGORITHM

✧ Decompose a number r into two parts: LowBits r_0 and HighBits r_1 .

Algorithm 36 *Decompose*(r)

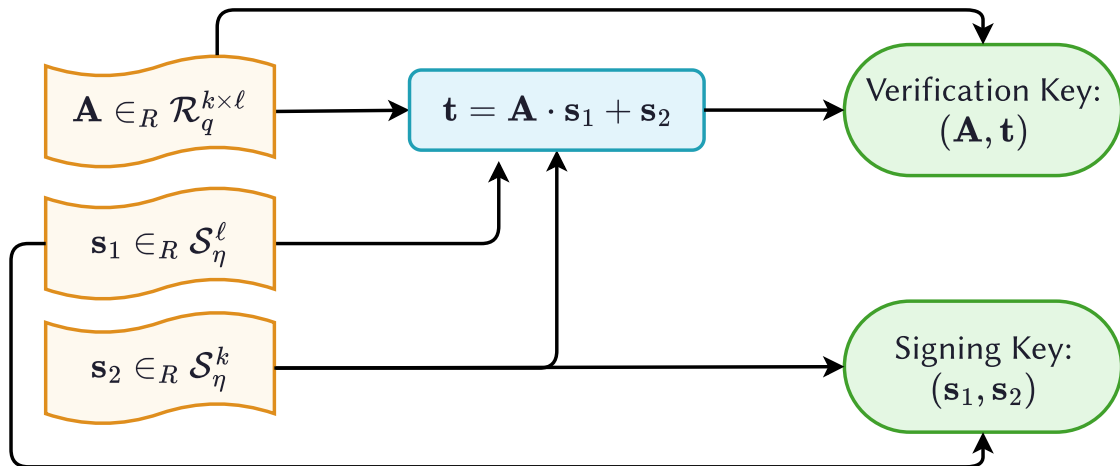
Decomposes r into (r_1, r_0) such that $r \equiv r_1(2\gamma_2) + r_0 \pmod{q}$.

Input: $r \in \mathbb{Z}_q$.

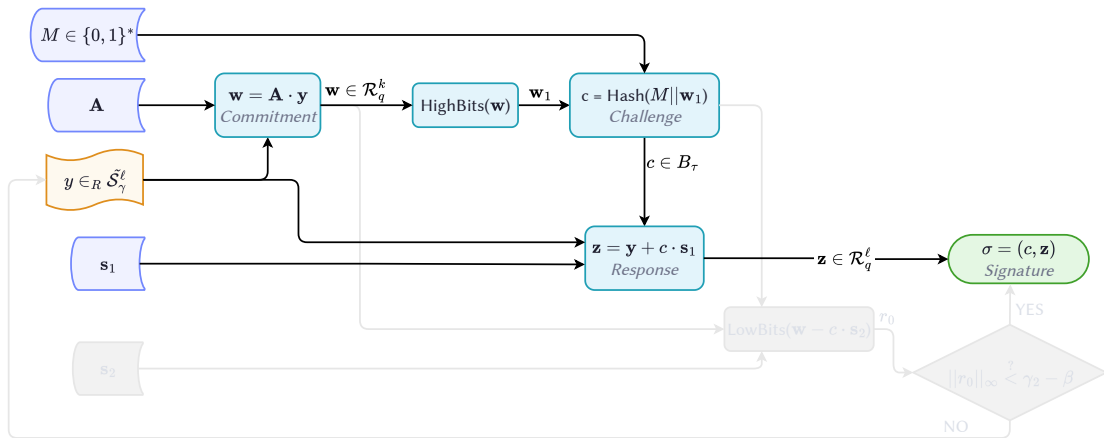
Output: Integers (r_1, r_0) .

```
1:  $r^+ \leftarrow r \bmod q$ 
2:  $r_0 \leftarrow r^+ \bmod^{\pm}(2\gamma_2)$ 
3: if  $r^+ - r_0 = q - 1$  then
4:    $r_1 \leftarrow 0$ 
5:    $r_0 \leftarrow r_0 - 1$ 
6: else  $r_1 \leftarrow (r^+ - r_0)/(2\gamma_2)$ 
7: end if
8: return  $(r_1, r_0)$ 
```

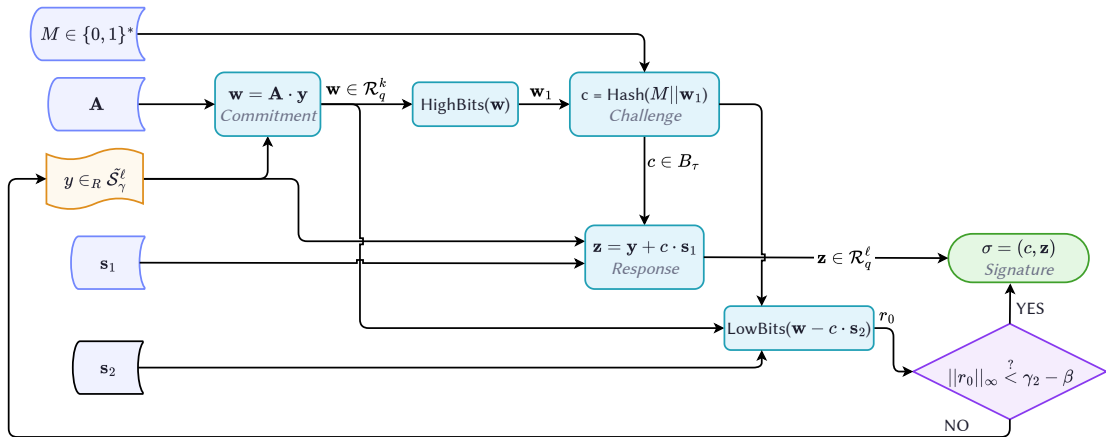
TOY DILITHIUM: KEY GENERATION



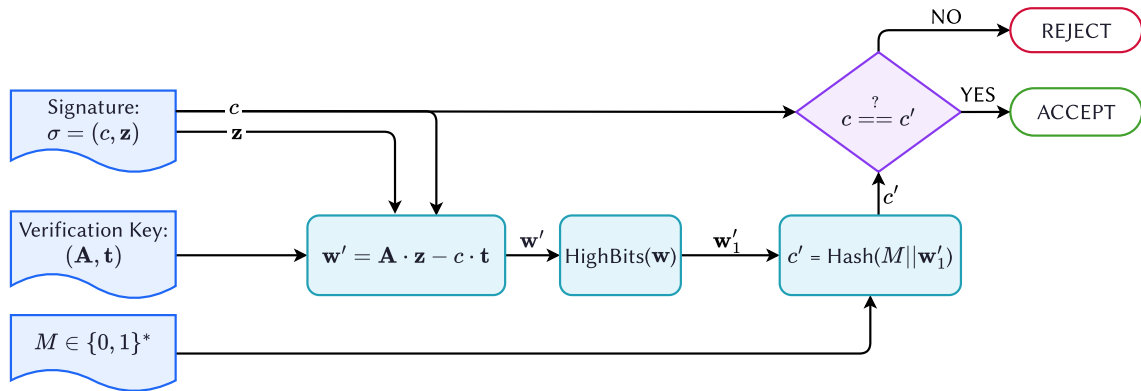
TOY DILITHIUM: SIGNATURE GENERATION



TOY DILITHIUM: SIGNATURE GENERATION



TOY DILITHIUM: SIGNATURE VERIFICATION



How Dilithium Guarantees the Authenticity of the Signature?

How It Works

✧ To prove that Dilithium works, we need to show that $\mathbf{w}'_1 = \mathbf{w}_1$.

✧ $\mathbf{w}_1 = \text{HighBits}(\mathbf{A} \cdot \mathbf{y}, 2\gamma_2)$, $\mathbf{w}'_1 = \text{HighBits}(\mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{t}, 2\gamma_2)$.

$$\begin{aligned}\mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{t} &= \mathbf{A} \cdot (\mathbf{y} + c \cdot \mathbf{s}_1) - c \cdot \mathbf{t} \\ &= \mathbf{A} \cdot \mathbf{y} + c(\mathbf{A} \cdot \mathbf{s}_1) - c \cdot \mathbf{t} \\ &= \mathbf{w} + c(\mathbf{A} \cdot \mathbf{s}_1 - \mathbf{t}) \\ &= \mathbf{w} + c(\mathbf{A} \cdot \mathbf{s}_1 - \mathbf{A} \cdot \mathbf{s}_1 - \mathbf{s}_2) \\ &= \mathbf{w} - c \cdot \mathbf{s}_2\end{aligned}$$

✧ The verifier has access to $\mathbf{A}$, $\mathbf{z}$, c and $\mathbf{t}$, but not $\mathbf{s}_2$.

✧ If $c \cdot \mathbf{s}_2$ is small enough, $\text{HighBits}(\mathbf{w} - c \cdot \mathbf{s}_2, 2\gamma_2) = \text{HighBits}(\mathbf{w}, 2\gamma_2) = \mathbf{w}_1$.

How It Works

✧ To prove that Dilithium works, we need to show that $\mathbf{w}'_1 = \mathbf{w}_1$.

✧ $\mathbf{w}_1 = \text{HighBits}(\mathbf{A} \cdot \mathbf{y}, 2\gamma_2)$, $\mathbf{w}'_1 = \text{HighBits}(\mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{t}, 2\gamma_2)$.

$$\begin{aligned}\mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{t} &= \mathbf{A} \cdot (\mathbf{y} + c \cdot \mathbf{s}_1) - c \cdot \mathbf{t} \\ &= \mathbf{A} \cdot \mathbf{y} + c(\mathbf{A} \cdot \mathbf{s}_1) - c \cdot \mathbf{t} \\ &= \mathbf{w} + c(\mathbf{A} \cdot \mathbf{s}_1 - \mathbf{t}) \\ &= \mathbf{w} + c(\mathbf{A} \cdot \mathbf{s}_1 - \mathbf{A} \cdot \mathbf{s}_1 - \mathbf{s}_2) \\ &= \mathbf{w} - c \cdot \mathbf{s}_2\end{aligned}$$

✧ The verifier has access to $\mathbf{A}$, $\mathbf{z}$, c and $\mathbf{t}$, but not $\mathbf{s}_2$.

✧ If $c \cdot \mathbf{s}_2$ is small enough, $\text{HighBits}(\mathbf{w} - c \cdot \mathbf{s}_2, 2\gamma_2) = \text{HighBits}(\mathbf{w}, 2\gamma_2) = \mathbf{w}_1$.

How It Works

✧ To prove that Dilithium works, we need to show that $\mathbf{w}'_1 = \mathbf{w}_1$.

✧ $\mathbf{w}_1 = \text{HighBits}(\mathbf{A} \cdot \mathbf{y}, 2\gamma_2)$, $\mathbf{w}'_1 = \text{HighBits}(\mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{t}, 2\gamma_2)$.

$$\begin{aligned}\mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{t} &= \mathbf{A} \cdot (\mathbf{y} + c \cdot \mathbf{s}_1) - c \cdot \mathbf{t} \\ &= \mathbf{A} \cdot \mathbf{y} + c(\mathbf{A} \cdot \mathbf{s}_1) - c \cdot \mathbf{t} \\ &= \mathbf{w} + c(\mathbf{A} \cdot \mathbf{s}_1 - \mathbf{t}) \\ &= \mathbf{w} + c(\mathbf{A} \cdot \mathbf{s}_1 - \mathbf{A} \cdot \mathbf{s}_1 - \mathbf{s}_2) \\ &= \mathbf{w} - c \cdot \mathbf{s}_2\end{aligned}$$

✧ The verifier has access to $\mathbf{A}$, $\mathbf{z}$, c and $\mathbf{t}$, but not $\mathbf{s}_2$.

✧ If $c \cdot \mathbf{s}_2$ is small enough, $\text{HighBits}(\mathbf{w} - c \cdot \mathbf{s}_2, 2\gamma_2) = \text{HighBits}(\mathbf{w}, 2\gamma_2) = \mathbf{w}_1$.

How much is $c \cdot s_2$ small enough?

- ✧ We want to add $c \cdot s_2$ to $\mathbf{w} - c \cdot s_2$ to get $\mathbf{w}$, but without changing the HighBits of $\mathbf{w} - c \cdot s_2$.
- ✧ If $c \cdot s_2$ is not small enough, it will change the HighBits of $\mathbf{w} - c \cdot s_2$ by ± 1 .
- ✧ Small enough:

Condition for $\mathbf{w} - c \cdot s_2$ (or Difference to Words)

$$\triangleright \left. \begin{array}{l} \|c\|_{\infty} \leq 1 \\ \|s_2\|_{\infty} \leq \eta \end{array} \right\} \Rightarrow \|c \cdot s_2\|_{\infty} \leq \underbrace{\eta + \eta + \dots + \eta}_{\tau} = \tau\eta = \beta$$

$$\triangleright \|\text{LowBits}(\mathbf{w}, 2\gamma_2)\|_{\infty} < \gamma_2$$

$$\triangleright \Rightarrow \|\text{LowBits}(\mathbf{w} - c \cdot s_2, 2\gamma_2)\|_{\infty} < \gamma_2 - \beta$$

How much is $c \cdot s_2$ small enough?

- ✧ We want to add $c \cdot s_2$ to $\mathbf{w} - c \cdot s_2$ to get $\mathbf{w}$, but without changing the HighBits of $\mathbf{w} - c \cdot s_2$.
- ✧ If $c \cdot s_2$ is not small enough, it will change the HighBits of $\mathbf{w} - c \cdot s_2$ by ± 1 .
- ✧ Small enough:

Condition for $\mathbf{w} - c \cdot s_2$ (for Dilithium to Work)

- ▷
$$\left. \begin{array}{l} \|c\|_{\infty} \leq 1 \\ \|s_2\|_{\infty} \leq \eta \end{array} \right\} \Rightarrow \|c \cdot s_2\|_{\infty} \leq \underbrace{\eta + \eta + \dots + \eta}_{\tau} = \tau\eta = \beta$$
- ▷ $\|\text{LowBits}(\mathbf{w}, 2\gamma_2)\|_{\infty} < \gamma_2$
- ▷ $\Rightarrow \|\text{LowBits}(\mathbf{w} - c \cdot s_2, 2\gamma_2)\|_{\infty} < \gamma_2 - \beta$

PROBLEMS OF TOY DILITHIUM AND REQUIRED OPTIMIZATIONS

Large Size of A

A is a matrix of size $k \times \ell$, each element is a polynomial with $n = 256$ terms, each 23 bits.

Solution: Using a public seed ρ like ExpandA in ML-KEM.

Large Size of s_1 and s_2

s_1 and s_2 are vectors of k and ℓ polynomials with $n = 256$ terms, each 23 bits.

Solution: Using a secret seed ρ' .

PROBLEMS OF TOY DILITHIUM AND REQUIRED OPTIMIZATIONS

Large Size of A

A is a matrix of size $k \times \ell$, each element is a polynomial with $n = 256$ terms, each 23 bits.

Solution: Using a public seed ρ like ExpandA in ML-KEM.

Large Size of s_1 and s_2

s_1 and s_2 are vectors of k and ℓ polynomials with $n = 256$ terms, each 23 bits.

Solution: Using a secret seed ρ' .

PROBLEMS OF TOY DILITHIUM AND REQUIRED OPTIMIZATIONS

Large Size of $\mathbf{t}$

$\mathbf{t}$ is a vector of size k . Elements are polynomial with $n = 256$ terms, each 23 bits.

Solution: Send only d higher bits of $\mathbf{t} \rightarrow$ Verifier cannot compute $\text{HighBits}(\mathbf{A} \cdot \mathbf{z} - \mathbf{c} \cdot \mathbf{t}) \rightarrow$ Using *hint bits*.

✧ Decompose $\mathbf{t}$ into $\mathbf{t}_1$ and $\mathbf{t}_0$: $\mathbf{t} = \mathbf{t}_1 \cdot 2^d + \mathbf{t}_0$. Only send $\mathbf{t}_1$ to the verifier.

✧ The verifier knows $\mathbf{t}_1$ and $d \rightarrow$ Can compute:

$$\mathbf{A} \cdot \mathbf{z} - \mathbf{c} \cdot \mathbf{t}_1 \cdot 2^d = \mathbf{A} \cdot \mathbf{z} - \mathbf{c}(\mathbf{t} - \mathbf{t}_0) = \mathbf{A} \cdot \mathbf{z} - \mathbf{c} \cdot \mathbf{t} + \mathbf{c} \cdot \mathbf{t}_0 = \mathbf{w} - \overbrace{\mathbf{c} \cdot \mathbf{s}_2 + \mathbf{c} \cdot \mathbf{t}_0}^{\text{may be large}}$$

PROBLEMS OF TOY DILITHIUM AND REQUIRED OPTIMIZATIONS

Large Size of $\mathbf{t}$

$\mathbf{t}$ is a vector of size k . Elements are polynomial with $n = 256$ terms, each 23 bits.

Solution: Send only d higher bits of $\mathbf{t} \rightarrow$ Verifier cannot compute $\text{HighBits}(\mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{t}) \rightarrow$ Using *hint bits*.

✧ Decompose $\mathbf{t}$ into $\mathbf{t}_1$ and $\mathbf{t}_0$: $\mathbf{t} = \mathbf{t}_1 \cdot 2^d + \mathbf{t}_0$. Only send $\mathbf{t}_1$ to the verifier.

✧ The verifier knows $\mathbf{t}_1$ and $d \rightarrow$ Can compute:

$$\mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{t}_1 \cdot 2^d = \mathbf{A} \cdot \mathbf{z} - c(\mathbf{t} - \mathbf{t}_0) = \mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{t} + c \cdot \mathbf{t}_0 = \mathbf{w} - \overbrace{c \cdot \mathbf{s}_2 + c \cdot \mathbf{t}_0}^{\text{may be large}}$$

PROBLEMS OF TOY DILITHIUM AND REQUIRED OPTIMIZATIONS

✧ If $\| -c \cdot \mathbf{t}_0 \|_\infty < \gamma_2 \rightarrow$ Adding $-c \cdot \mathbf{t}_0$ to $\mathbf{w} - c \cdot \mathbf{s}_2 + c \cdot \mathbf{t}_0$ will change HighBits by 0, 1, or -1.

Condition for $-c \cdot \mathbf{t}_0$ (For Efficiency of Dilithium)

If $\| -c \cdot \mathbf{t}_0 \|_\infty < \gamma_2$:

Use **hint bits** to notify the verifier to adjust the HighBits of $\mathbf{w}$ by 1, or -1.

✧ For efficiency, the number of 1's in the hint vector is at most ω to reduce the required size.

PROBLEMS OF TOY DILITHIUM AND REQUIRED OPTIMIZATIONS

Condition for $\mathbf{z}$ (For Security Reasons)

$\mathbf{y} \in \tilde{S}_{\gamma_1}^\ell$, $c \in B_\tau$, $\mathbf{s}_1 \in S_\eta^\ell$ and $\mathbf{z} = \mathbf{y} + c \cdot \mathbf{s}_1 \Rightarrow \|\mathbf{z}\|_\infty \leq \gamma_1 + \beta$:

✧ If $\|\mathbf{z}\|_\infty = \gamma_1 + \beta \Rightarrow c \cdot \mathbf{s}_1 = \beta$

✧ If $\|\mathbf{z}\|_\infty = \gamma_1 + \beta - \epsilon$, $\epsilon < 2\beta \Rightarrow c \cdot \mathbf{s}_1 \in \{\beta, \beta - 1, \dots, \beta - \epsilon\}$

✧ To avoid revealing information about $c \cdot \mathbf{s}_1$ and thus $\mathbf{s}_1 \rightarrow \|\mathbf{z}\|_\infty = \gamma_1 - \beta$.

Large Number of Random Bits

A large number of random bits are needed $\rightarrow$ Generate from a secret seed and a counter.

Repeating Computation of $H(\cdot)$

Each time we need to compute $H(M) \rightarrow$ Compute $\mu = H(M)$ at first and one time.

PROBLEMS OF TOY DILITHIUM AND REQUIRED OPTIMIZATIONS

Condition for $\mathbf{z}$ (For Security Reasons)

$\mathbf{y} \in \tilde{S}_{\gamma_1}^\ell$, $c \in B_\tau$, $\mathbf{s}_1 \in S_\eta^\ell$ and $\mathbf{z} = \mathbf{y} + c \cdot \mathbf{s}_1 \Rightarrow \|\mathbf{z}\|_\infty \leq \gamma_1 + \beta$:

✧ If $\|\mathbf{z}\|_\infty = \gamma_1 + \beta \Rightarrow c \cdot \mathbf{s}_1 = \beta$

✧ If $\|\mathbf{z}\|_\infty = \gamma_1 + \beta - \epsilon$, $\epsilon < 2\beta \Rightarrow c \cdot \mathbf{s}_1 \in \{\beta, \beta - 1, \dots, \beta - \epsilon\}$

✧ To avoid revealing information about $c \cdot \mathbf{s}_1$ and thus $\mathbf{s}_1 \rightarrow \|\mathbf{z}\|_\infty = \gamma_1 - \beta$.

Large Number of Random Bits

A large number of random bits are needed $\rightarrow$ Generate from a secret seed and a counter.

Repeating Computation of $H(\cdot)$

Each time we need to compute $H(M) \rightarrow$ Compute $\mu = H(M)$ at first and one time.

PROBLEMS OF TOY DILITHIUM AND REQUIRED OPTIMIZATIONS

Condition for $\mathbf{z}$ (For Security Reasons)

$\mathbf{y} \in \tilde{S}_{\gamma_1}^\ell$, $c \in B_\tau$, $\mathbf{s}_1 \in S_\eta^\ell$ and $\mathbf{z} = \mathbf{y} + c \cdot \mathbf{s}_1 \Rightarrow \|\mathbf{z}\|_\infty \leq \gamma_1 + \beta$:

✧ If $\|\mathbf{z}\|_\infty = \gamma_1 + \beta \Rightarrow c \cdot \mathbf{s}_1 = \beta$

✧ If $\|\mathbf{z}\|_\infty = \gamma_1 + \beta - \epsilon$, $\epsilon < 2\beta \Rightarrow c \cdot \mathbf{s}_1 \in \{\beta, \beta - 1, \dots, \beta - \epsilon\}$

✧ To avoid revealing information about $c \cdot \mathbf{s}_1$ and thus $\mathbf{s}_1 \rightarrow \|\mathbf{z}\|_\infty = \gamma_1 - \beta$.

Large Number of Random Bits

A large number of random bits are needed $\rightarrow$ Generate from a secret seed and a counter.

Repeating Computation of $H(M)$

Each time we need to compute $H(M) \rightarrow$ Compute $\mu = H(M)$ at first and one time.

DILITHIUM COMPONENTS: POWER2ROUND

- ✧ **Power2Round:** A bit-wise breaking up a number r into two parts r_1 (higher d bits) and r_0 (rest of the bits):

Power2Round

$$r = r_1 \cdot 2^d + r_0,$$

$$-2^{d-1} < r_0 \leq 2^{d-1}, \quad 0 \leq r_1 \leq \frac{q-1}{2^d}$$

- ✧ The only difference between Power2Round and Decompose is that Power2Round allows r_1 to be equal to $\frac{q-1}{2^d}$ (it doesn't have mod q).

DILITHIUM COMPONENTS

ExpandMask

Deterministically generates randomness in every sample of Dilithium from a seed ρ'' and a counter κ .

SampleInBall

A function to sample a polynomial $c \in \mathcal{R}_q$ with coefficients $\in \{-1, 0, 1\}$, s.t. number of ± 1 is exactly τ ($\text{HammingWeight}(c) = \tau$).

Example of SampleInBall

SampleInBall(A 2λ -bit number) = $\overbrace{-101-10100-100 \dots 0}^{256 \text{ bits}}$.

Number of ± 1 (red bits) is exactly τ .

DILITHIUM COMPONENTS

ExpandMask

Deterministically generates randomness in every sample of Dilithium from a seed ρ'' and a counter κ .

SampleInBall

A function to sample a polynomial $c \in \mathcal{R}_q$ with coefficients $\in \{-1, 0, 1\}$, s.t. number of ± 1 is exactly τ ($\text{HammingWeight}(c) = \tau$).

Example of SampleInBall

SampleInBall(A 2λ -bit number) = $\overbrace{-101-10100-100 \dots 0}^{256 \text{ bits}}$.

Number of ± 1 (red bits) is exactly τ .

DILITHIUM COMPONENTS: HINT

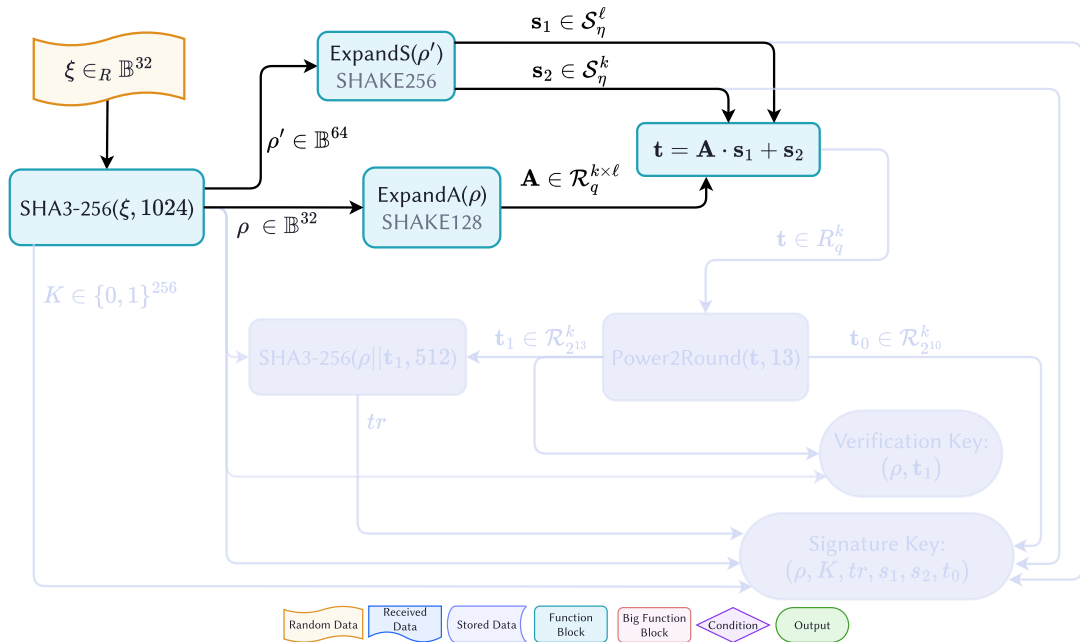
Hint

- ✧ **MakeHint:** Generates **one bit** of hint for each coefficient $\rightarrow$ Hint vector is a matrix of size $k \times n$.
- ✧ **UseHint:** Computes the HighBits of $\mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{t}_1 \cdot 2^d$ by the verifier using the hint bits (hint vector).

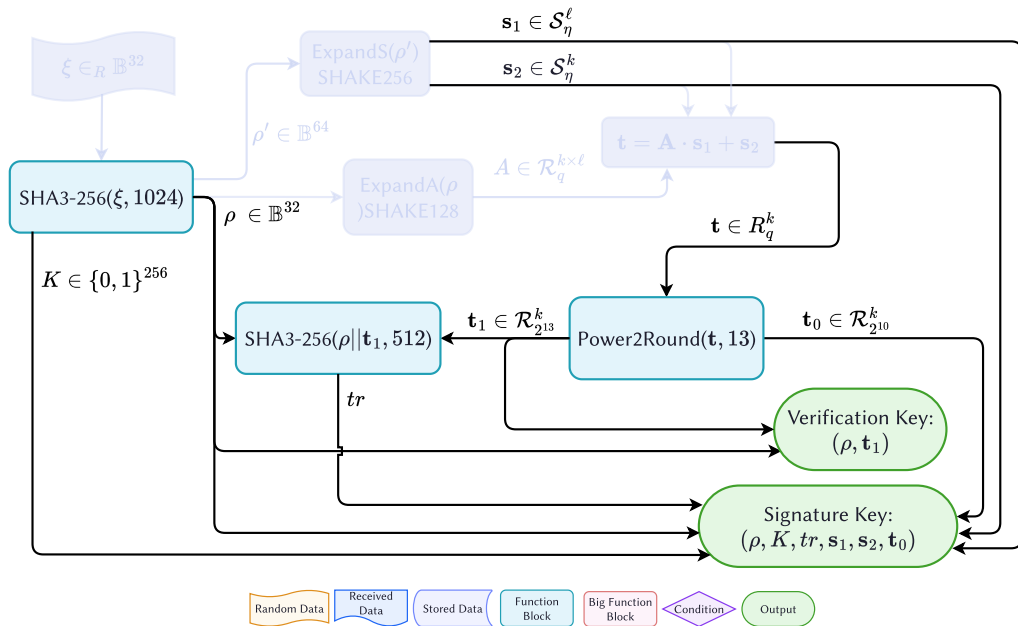
Condition for Hint Vector (For Efficiency of Dilithium)

The number of 1's in the hint vector is at most ω .

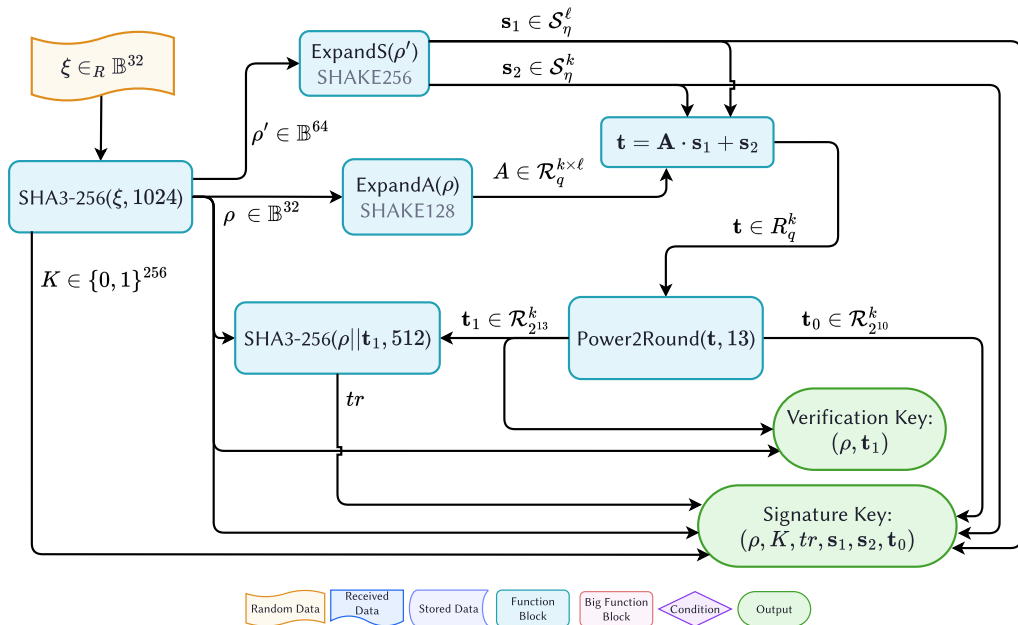
DILITHIUM: KEY GENERATION



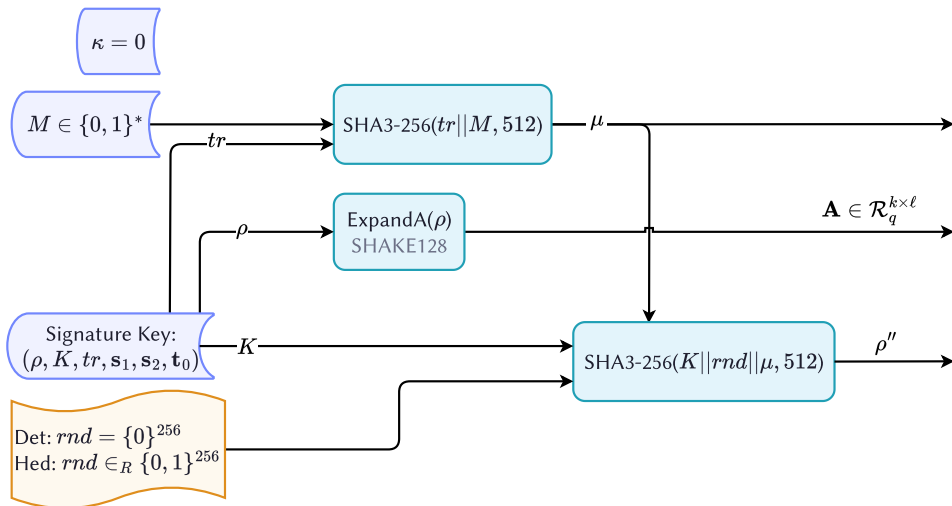
DILITHIUM: KEY GENERATION



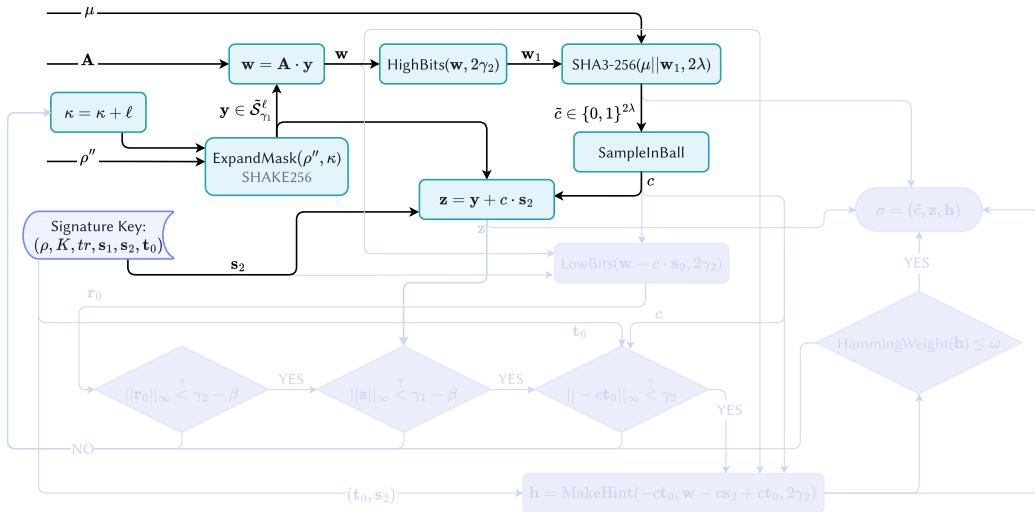
DILITHIUM: KEY GENERATION



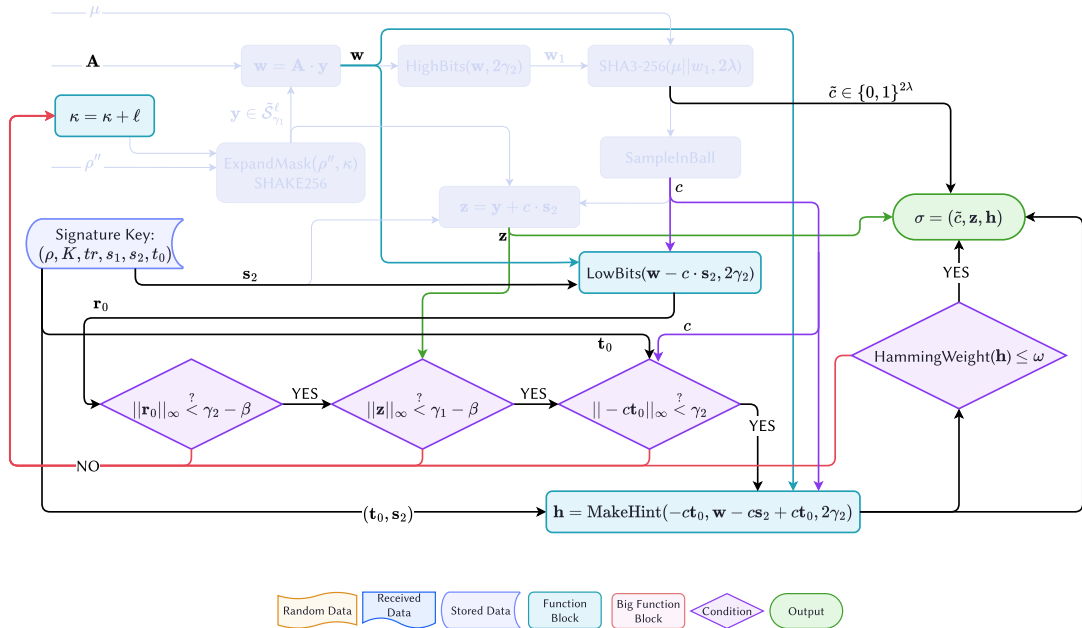
DILITHIUM: SIGNATURE GENERATION - PART 1



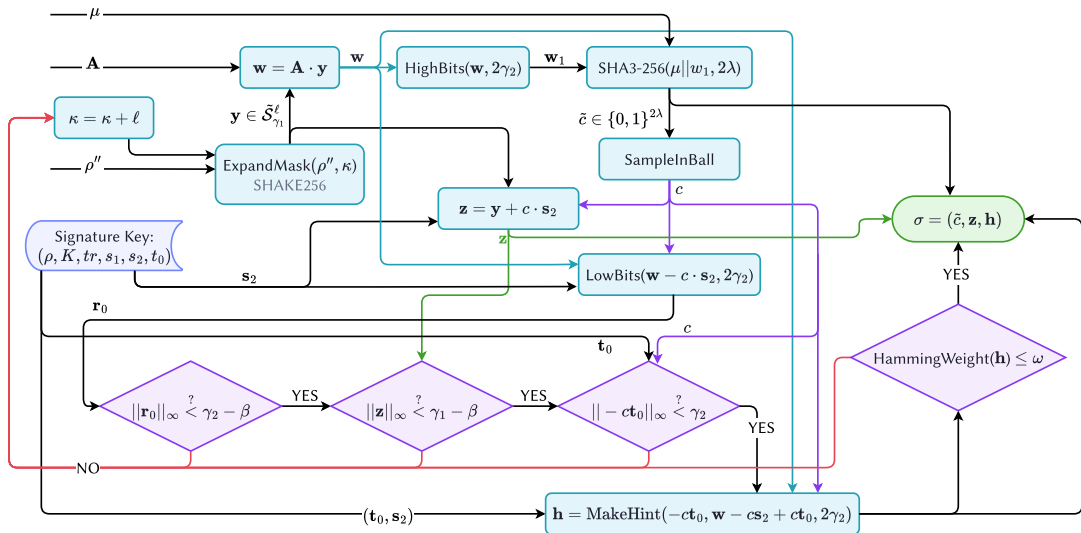
DILITHIUM: SIGNATURE GENERATION - PART 2



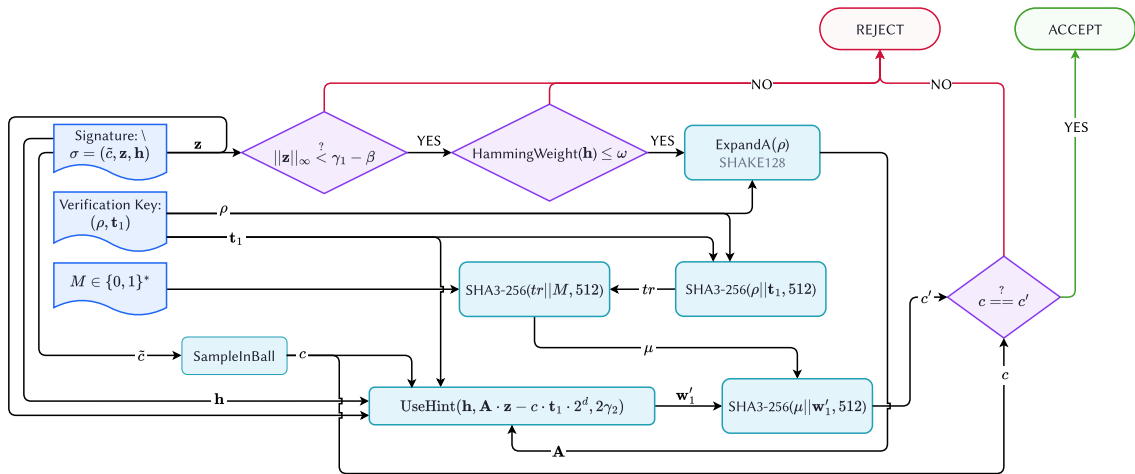
DILITHIUM: SIGNATURE GENERATION - PART 2



DILITHIUM: SIGNATURE GENERATION - PART 2



DILITHIUM: SIGNATURE VERIFICATION



✧ A pseudo-random generator of an element of B_τ using H (SHAKE256) of a seed ρ .

Algorithm 29 SampleInBall(ρ)

Samples a polynomial $c \in R$ with coefficients from $\{-1, 0, 1\}$ and Hamming weight $\tau \leq 64$.

Input: A seed $\rho \in \mathbb{B}^{\lambda/4}$

Output: A polynomial c in R .

```

1:  $c \leftarrow 0$ 
2:  $\text{ctx} \leftarrow \text{H.Init}()$ 
3:  $\text{ctx} \leftarrow \text{H.Absorb}(\text{ctx}, \rho)$ 
4:  $(\text{ctx}, s) \leftarrow \text{H.Squeeze}(\text{ctx}, 8)$ 
5:  $h \leftarrow \text{BytesToBits}(s)$   $\triangleright h$  is a bit string of length 64
6: for  $i$  from  $256 - \tau$  to  $255$  do
7:    $(\text{ctx}, j) \leftarrow \text{H.Squeeze}(\text{ctx}, 1)$ 
8:   while  $j > i$  do  $\triangleright$  rejection sampling in  $\{0, \dots, i\}$ 
9:      $(\text{ctx}, j) \leftarrow \text{H.Squeeze}(\text{ctx}, 1)$ 
10:  end while  $\triangleright j$  is a pseudorandom byte that is  $\leq i$ 
11:    $c_i \leftarrow c_j$ 
12:    $c_j \leftarrow (-1)^{h[i+\tau-256]}$ 
13: end for
14: return  $c$ 
    
```

Algorithm 39 `MakeHint`(z, r)

Computes hint bit indicating whether adding z to r alters the high bits of r .

Input: $z, r \in \mathbb{Z}_q$.

Output: Boolean.

- 1: $r_1 \leftarrow \text{HighBits}(r)$
 - 2: $v_1 \leftarrow \text{HighBits}(r + z)$
 - 3: **return** $[[r_1 \neq v_1]]$
-

Algorithm 40 `UseHint`(h, r)

Returns the high bits of r adjusted according to hint h .

Input: Boolean $h, r \in \mathbb{Z}_q$.

Output: $r_1 \in \mathbb{Z}$ with $0 \leq r_1 \leq \frac{q-1}{2\gamma_2}$.

- 1: $m \leftarrow (q - 1)/(2\gamma_2)$
 - 2: $(r_1, r_0) \leftarrow \text{Decompose}(r)$
 - 3: **if** $h = 1$ and $r_0 > 0$ **return** $(r_1 + 1) \bmod m$
 - 4: **if** $h = 1$ and $r_0 \leq 0$ **return** $(r_1 - 1) \bmod m$
 - 5: **return** r_1
-

RANDOM SIGNATURE GENERATION IN DILITHIUM

- ✧ Dilithium has the ability to generate **fresh** signature each time; An adversary cannot generate a new signature for a message that has already been signed by the signer.
- ✧ It has two modes:
 - ▷ Deterministic: The same message will always produce the same signature.
 - ▷ Hedged (randomized): The same message will produce different signatures each time.
- ✧ The signer should not use the deterministic variant of ML-DSA on platforms where side-channel attacks or fault attacks are a concern (*IETF*).
- ✧ We can use the deterministic variant of ML-DSA where:
 - ▷ The signer is not concerned about side-channel attacks.
 - ▷ The same message should always produce the same signature (like logging).
 - ▷ The signer does not have a reliable TRNG.

RANDOM SIGNATURE GENERATION IN DILITHIUM

- ✧ Dilithium has the ability to generate **fresh** signature each time; An adversary cannot generate a new signature for a message that has already been signed by the signer.
- ✧ It has two modes:
 - ▷ Deterministic: The same message will always produce the same signature.
 - ▷ Hedged (randomized): The same message will produce different signatures each time.
- ✧ The signer should not use the deterministic variant of ML-DSA on platforms where side-channel attacks or fault attacks are a concern (*IETF*).
- ✧ We can use the deterministic variant of ML-DSA where:
 - ▷ The signer is not concerned about side-channel attacks.
 - ▷ The same message should always produce the same signature (like logging).
 - ▷ The signer does not have a reliable TRNG.

RANDOM SIGNATURE GENERATION IN DILITHIUM

- ✧ Dilithium has the ability to generate **fresh** signature each time; An adversary cannot generate a new signature for a message that has already been signed by the signer.
- ✧ It has two modes:
 - ▷ Deterministic: The same message will always produce the same signature.
 - ▷ Hedged (randomized): The same message will produce different signatures each time.
- ✧ The signer should not use the deterministic variant of ML-DSA on platforms where side-channel attacks or fault attacks are a concern (*IETF*).
- ✧ We can use the deterministic variant of ML-DSA where:
 - ▷ The signer is not concerned about side-channel attacks.
 - ▷ The same message should always produce the same signature (like logging).
 - ▷ The signer does not have a reliable TRNG.

RANDOM SIGNATURE GENERATION IN DILITHIUM

- ✧ Dilithium has the ability to generate **fresh** signature each time; An adversary cannot generate a new signature for a message that has already been signed by the signer.
- ✧ It has two modes:
 - ▷ Deterministic: The same message will always produce the same signature.
 - ▷ Hedged (randomized): The same message will produce different signatures each time.
- ✧ The signer should not use the deterministic variant of ML-DSA on platforms where side-channel attacks or fault attacks are a concern (*IETF*).
- ✧ We can use the deterministic variant of ML-DSA where:
 - ▷ The signer is not concerned about side-channel attacks.
 - ▷ The same message should always produce the same signature (like logging).
 - ▷ The signer does not have a reliable TRNG.

SECURITY OF DILITHIUM

✧ **MLWE:** The attacker cannot find the secrets $\mathbf{s}_1$ and $\mathbf{s}_2$ given the public key $(\mathbf{t}, \mathbf{A})$ (or $(\rho, \mathbf{t}_1)$) from $\mathbf{t} = \mathbf{A} \cdot \mathbf{s}_1 + \mathbf{s}_2$ (security in KeyGen).

✧ **IMISIS:** Every parameters in signature generation is forgeable except $\mathbf{z} = \mathbf{y} + c \cdot \mathbf{s}_1$:

▷ Select a Random $\mathbf{w}_1 \in \mathcal{R}_q^k$, s.t. $\|\mathbf{w}_1\|_\infty < \frac{q-1}{2\gamma_2}$.

▷ Compute $\mu = H(M) \rightarrow \tilde{c} = H(\mu \parallel \mathbf{w}_1) \rightarrow c = \text{SampleInBall}(\tilde{c})$.

✧ To find $\mathbf{z}$:

$$\mathbf{w}_1 = \text{HighBits}(\mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{t}, 2\gamma_2) \Rightarrow \mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{t} = 2\gamma_2 \mathbf{w}_1 + \mathbf{w}_0 \Rightarrow \mathbf{A} \cdot \mathbf{z} - \mathbf{w}_0 = c \cdot \mathbf{t} + 2\gamma_2 \mathbf{w}_1$$

✧ Equivalently:

$$[\mathbf{A} | I_k] \begin{bmatrix} \mathbf{z} \\ -\mathbf{w}_0 \end{bmatrix} = c \cdot \mathbf{t} + 2\gamma_2 \mathbf{w}_1$$

✧ $\mathbf{z}$ and $\mathbf{w}_0$ are small enough $\rightarrow$ An instance of ISIS $\rightarrow$ The attacker cannot find $\mathbf{z}$.

SECURITY OF DILITHIUM

✧ **MLWE:** The attacker cannot find the secrets $\mathbf{s}_1$ and $\mathbf{s}_2$ given the public key $(\mathbf{t}, \mathbf{A})$ (or $(\rho, \mathbf{t}_1)$) from $\mathbf{t} = \mathbf{A} \cdot \mathbf{s}_1 + \mathbf{s}_2$ (security in KeyGen).

✧ **IMISIS:** Every parameters in signature generation is forgeable except $\mathbf{z} = \mathbf{y} + c \cdot \mathbf{s}_1$:

▷ Select a Random $\mathbf{w}_1 \in \mathcal{R}_q^k$, s.t. $\|\mathbf{w}_1\|_\infty < \frac{q-1}{2\gamma_2}$.

▷ Compute $\mu = H(M) \rightarrow \tilde{c} = H(\mu \parallel \mathbf{w}_1) \rightarrow c = \text{SampleInBall}(\tilde{c})$.

✧ To find $\mathbf{z}$:

$$\mathbf{w}_1 = \text{HighBits}(\mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{t}, 2\gamma_2) \Rightarrow \mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{t} = 2\gamma_2 \mathbf{w}_1 + \mathbf{w}_0 \Rightarrow \mathbf{A} \cdot \mathbf{z} - \mathbf{w}_0 = c \cdot \mathbf{t} + 2\gamma_2 \mathbf{w}_1$$

✧ Equivalently:

$$[\mathbf{A} | I_k] \begin{bmatrix} \mathbf{z} \\ -\mathbf{w}_0 \end{bmatrix} = c \cdot \mathbf{t} + 2\gamma_2 \mathbf{w}_1$$

✧ $\mathbf{z}$ and $\mathbf{w}_0$ are small enough $\rightarrow$ An instance of IMISIS $\rightarrow$ The attacker cannot find $\mathbf{z}$.

SECURITY OF DILITHIUM

✧ **MLWE:** The attacker cannot find the secrets $\mathbf{s}_1$ and $\mathbf{s}_2$ given the public key $(\mathbf{t}, \mathbf{A})$ (or $(\rho, \mathbf{t}_1)$) from $\mathbf{t} = \mathbf{A} \cdot \mathbf{s}_1 + \mathbf{s}_2$ (security in KeyGen).

✧ **IMISIS:** Every parameters in signature generation is forgeable except $\mathbf{z} = \mathbf{y} + c \cdot \mathbf{s}_1$:

▷ Select a Random $\mathbf{w}_1 \in \mathcal{R}_q^k$, s.t. $\|\mathbf{w}_1\|_\infty < \frac{q-1}{2\gamma_2}$.

▷ Compute $\mu = H(M) \rightarrow \tilde{c} = H(\mu \parallel \mathbf{w}_1) \rightarrow c = \text{SampleInBall}(\tilde{c})$.

✧ To find $\mathbf{z}$:

$$\mathbf{w}_1 = \text{HighBits}(\mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{t}, 2\gamma_2) \Rightarrow \mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{t} = 2\gamma_2 \mathbf{w}_1 + \mathbf{w}_0 \Rightarrow \mathbf{A} \cdot \mathbf{z} - \mathbf{w}_0 = c \cdot \mathbf{t} + 2\gamma_2 \mathbf{w}_1$$

✧ Equivalently:

$$[\mathbf{A} | I_k] \begin{bmatrix} \mathbf{z} \\ -\mathbf{w}_0 \end{bmatrix} = c \cdot \mathbf{t} + 2\gamma_2 \mathbf{w}_1$$

✧ $\mathbf{z}$ and $\mathbf{w}_0$ are small enough $\rightarrow$ An instance of IMISIS $\rightarrow$ The attacker cannot find $\mathbf{z}$.

Thank you!