

TCVN

TIÊU CHUẨN QUỐC GIA

DỰ THẢO

TCVN XXXXX:2026

**LƯỢC ĐỒ CHỮ KÝ SỐ DỰA TRÊN LƯỚI MÔ-ĐUN
ML-DSA**

HÀ NỘI - 2026

MỤC LỤC

1. Phạm vi áp dụng	5
2. Tài liệu viện dẫn	5
3. Thuật ngữ và định nghĩa	5
4. Ký hiệu, chữ viết tắt và ký hiệu toán học	8
4.1. Quy ước ký hiệu	10
4.1.1. Vành	10
4.1.2. Vectơ và ma trận	11
4.2. Biểu diễn NTT	11
5. Tổng quan về lược đồ chữ ký số ML-DSA	12
5.1. Các tính chất an toàn	12
5.2. Các giả thiết tính toán	12
5.3. Cấu trúc ML-DSA	12
5.4. Ký có dự phòng và Ký tắt định	14
5.5. Sử dụng chữ ký số	15
5.6. Các yêu cầu bổ sung	15
5.6.1. Sinh các giá trị ngẫu nhiên	15
5.6.2. Kiểm tra độ dài khóa công khai và chữ ký	16
5.6.3. Các giá trị trung gian	16
5.6.4. Không sử dụng số học dấu phẩy động	17
5.7. Sử dụng mật mã đối xứng	17
6. Các bộ tham số	18
7. Các hàm bên ngoài	20
7.1. Tạo khóa ML-DSA	21
7.2. Ký số ML-DSA	21
7.3. Xác minh ML-DSA	22
7.4. ML-DSA dạng băm trước	22
7.4.1. Ký và Xác minh HashML-DSA	23
8. Các hàm bên trong	26

8.1.	Tạo khóa ML-DSA (bên trong)	26
8.2.	Quá trình ký ML-DSA (bên trong)	27
8.3.	Xác minh ML-DSA (Bên trong)	30
9.	Các hàm hỗ trợ	32
9.1.	Chuyển đổi giữa các kiểu dữ liệu	32
9.2.	Mã hóa khóa và chữ ký ML-DSA	37
9.3.	Lấy mẫu giả ngẫu nhiên	40
9.4.	Các bit bậc cao, bậc thấp và gợi ý	44
9.5.	NTT và $NTT - 1$	46
9.6.	Số học dưới phép biến đổi NTT	49
	PHỤ LỤC A – Phép nhân Montgomery	51
	PHỤ LỤC B – Mảng Zetas	53
	PHỤ LỤC C — Các giới hạn vòng lặp	54
	PHỤ LỤC D — Những khác biệt so với bản hồ sơ đệ trình CRYSTALS-Dilithium	56
	Thư mục tài liệu tham khảo	58

Lời nói đầu

Tiêu chuẩn *TCVN XXXXX:2026* *Lược đồ chữ ký số dựa trên lưới mô-đun ML-DSA* do Viện Khoa học - Công nghệ mật mã, Ban Cơ yếu Chính phủ biên soạn, Ban Cơ yếu Chính phủ đề nghị, Ủy ban Tiêu chuẩn Đo lường Chất lượng Quốc gia thẩm định, Bộ Khoa học và Công nghệ công bố.

Tiêu chuẩn *TCVN XXXXX:2026* *Lược đồ chữ ký số dựa trên lưới mô-đun ML-DSA* được xây dựng trên cơ sở chấp nhận hoàn toàn tương đương với tiêu chuẩn FIPS 204 (2024), *Module-Lattice-Based Digital Signature Standard* của Viện Tiêu chuẩn và Công nghệ Quốc gia Hoa Kỳ (NIST).

Công nghệ thông tin – Các kỹ thuật an toàn – Lược đồ chữ ký số dựa trên lưới mô-đun ML-DSA

Information technology – Security techniques – Module-Lattice-Based Digital Signature Standard

1. Phạm vi áp dụng

Tiêu chuẩn này quy định lược đồ chữ ký số dựa trên lưới mô-đun ML-DSA (dựa trên bài toán học có lỗi trên mô-đun [4]), gồm các thuật toán sinh khóa, ký và xác minh; biến thể ký trước băm HashML-DSA; ba bộ tham số ML-DSA-44, ML-DSA-65, ML-DSA-87 và các hàm hỗ trợ. Thuật toán này được sử dụng để đảm bảo tính xác thực và toàn vẹn dữ liệu trong các ứng dụng mật mã và bảo mật thông tin.

ML-DSA được tin rằng là an toàn, ngay cả trước các bên đối kháng sở hữu một máy tính lượng tử quy mô lớn có khả năng chịu lỗi. Cụ thể, ML-DSA được tin rằng đạt được tính không thể giả mạo mạnh, hàm ý rằng lược đồ này có thể được sử dụng để phát hiện các sửa đổi trái phép đối với dữ liệu và để xác thực danh tính của bên ký (người gắn với việc sở hữu khóa bí mật). Ngoài ra, một chữ ký được tạo bởi lược đồ này có thể được sử dụng làm bằng chứng khi chứng minh với bên thứ ba rằng chữ ký đó thực sự được tạo ra bởi đúng bên ký đã được tuyên bố; tính chất này được gọi là tính không thể chối bỏ vì bên ký không thể dễ dàng chối bỏ chữ ký vào một thời điểm sau đó.

Tiêu chuẩn này áp dụng đối với các tổ chức, cá nhân thiết kế, phát triển, đánh giá và sử dụng hệ thống chữ ký số đảm bảo tính an toàn lâu dài trước các tấn công bằng máy tính lượng tử tiềm tàng.

2. Tài liệu viện dẫn

Các tài liệu viện dẫn sau rất cần thiết cho việc áp dụng tiêu chuẩn này. Đối với các tài liệu viện dẫn ghi năm công bố thì áp dụng phiên bản được nêu. Đối với các tài liệu viện dẫn không ghi năm công bố thì áp dụng phiên bản mới nhất, bao gồm cả các sửa đổi, bổ sung (nếu có).

- TCVN 12853:2020 Công nghệ thông tin – Các kỹ thuật an toàn – Bộ tạo bit ngẫu nhiên.
- TCVN 11816-3:2017 Công nghệ thông tin – Các kỹ thuật an toàn – Hàm băm – Phần 3: Hàm băm chuyên dụng.
- ISO/IEC 10118-3:2018 Information technology — Security techniques — Hash-functions — Part 3: Dedicated hash-functions

3. Thuật ngữ và định nghĩa

Tiêu chuẩn này sử dụng các thuật ngữ và định nghĩa sau:

3.1.

Chuỗi bit

Một dãy có thứ tự gồm các số 0 và 1.

3.2.

Byte

Một số nguyên thuộc tập hợp $\{0, \dots, 255\}$.

3.3.

Chuỗi byte

Một dãy byte có thứ tự.

3.4.

Chứng thư

Một tập hợp dữ liệu xác định duy nhất một khóa công khai có khóa bí mật tương ứng và một chủ sở hữu được phép sử dụng cặp khóa. Chứng thư chứa khóa công khai của chủ sở hữu và có thể chứa các thông tin khác, đồng thời được một tổ chức chứng thực (tức là một bên đáng tin cậy) ký số, qua đó gắn khóa công khai với chủ sở hữu.

3.5.

Tổ chức chứng thực (CA)

Thực thể trong hạ tầng khóa công khai (PKI) chịu trách nhiệm cấp chứng thư và yêu cầu tuân thủ chính sách PKI.

3.6.

Chữ ký số

Kết quả của một phép biến đổi mật mã đối với dữ liệu mà khi được triển khai đúng, cung cấp cơ chế để xác minh tính xác thực của nguồn gốc và tính toàn vẹn của dữ liệu, đồng thời bảo đảm tính chống chối bỏ của bên ký.

3.7.

Thực thể

Một cá nhân, tổ chức, thiết bị hoặc tiến trình. Được sử dụng thay thế cho thuật ngữ “bên”.

3.8.

Quy trình tương đương

Hai quy trình là tương đương nếu tạo ra cùng một đầu ra khi các giá trị đầu vào của mỗi quy trình là như nhau (dưới dạng tham số đầu vào, các giá trị được cung cấp trong quá trình thực hiện, hoặc cả hai).

3.9.

Hàm đầu ra mở rộng (XOF)

Một hàm trên các chuỗi bit, trong đó đầu ra có thể được mở rộng đến bất kỳ độ dài mong muốn nào. Các XOF được quy định trong tiêu chuẩn này (ví dụ: các hàm được quy định trong FIPS 202 [7]) được thiết kế để thỏa mãn các tính chất sau, miễn là độ dài đầu ra được chỉ định đủ lớn để ngăn chặn các tấn công tầm thường.

3.10.

Giá trị ngẫu nhiên mới

Một đầu ra chưa từng được sử dụng của một bộ sinh bit ngẫu nhiên.

3.11.**Hàm băm**

Một hàm trên các chuỗi bit có độ dài đầu ra cố định. Các hàm băm được quy định trong tiêu chuẩn này (chẳng hạn như các hàm được quy định trong FIPS 180 [8] và FIPS 202 [7]) được thiết kế để thỏa mãn các tính chất sau:

1. (Tính một chiều) Không khả thi về mặt tính toán để tìm một đầu vào bất kỳ ánh xạ tới một đầu ra mới bất kỳ được xác định trước.
2. (Tính kháng va chạm) Không khả thi về mặt tính toán để tìm hai đầu vào khác nhau bất kỳ ánh xạ tới cùng một đầu ra.

3.12.**Giá trị băm thông điệp**

Kết quả của việc áp dụng một hàm băm cho một thông điệp. Còn được gọi là giá trị băm.

3.13.**Chủ sở hữu**

Chủ sở hữu cặp khóa là thực thể được phép sử dụng khóa bí mật của một cặp khóa.

3.14.**Khóa bí mật**

Một khóa mật mã được sử dụng với một thuật toán mật mã bất đối xứng (khóa công khai). Khóa bí mật gắn duy nhất với chủ sở hữu và không được công khai. Khóa bí mật được sử dụng để tính một chữ ký số có thể được xác minh bằng khóa công khai tương ứng.

3.15.**Giả ngẫu nhiên**

Một quy trình hoặc dữ liệu do một quy trình tạo ra được gọi là giả ngẫu nhiên khi kết quả có tính tắt định nhưng vẫn ngẫu nhiên về mặt thực tiễn, miễn là hoạt động nội bộ của quy trình được che giấu khỏi sự quan sát. Đối với mục đích mật mã, “ngẫu nhiên về mặt thực tiễn” có nghĩa là “không thể phân biệt được về mặt tính toán với ngẫu nhiên trong giới hạn của độ mạnh an toàn dự kiến”.

3.16.**Khóa công khai**

Một khóa mật mã được sử dụng với một thuật toán mật mã bất đối xứng (khóa công khai) và được liên kết với một khóa bí mật. Khóa công khai gắn với một chủ sở hữu và có thể được công khai. Trong trường hợp chữ ký số, khóa công khai được sử dụng để xác minh một chữ ký số được tạo bằng khóa bí mật tương ứng

3.17.**Độ mạnh an toàn**

Một con số gắn với lượng công việc (tức là số phép toán) cần thiết để phá một thuật toán hoặc hệ thống mật mã.

3.18.

Mầm

Một chuỗi bit được sử dụng làm đầu vào cho một quy trình giả ngẫu nhiên.

3.19.**Sinh chữ ký**

Quá trình sử dụng một thuật toán chữ ký số và một khóa bí mật để tạo ra một chữ ký số trên dữ liệu.

3.20.**Xác minh chữ ký**

Quá trình sử dụng một thuật toán chữ ký số và một khóa công khai để xác minh một chữ ký số trên dữ liệu.

3.21.**Cấp độ an toàn**

Một con số gắn với độ mạnh an toàn của một thuật toán mật mã hậu lượng tử, theo quy định của NIST.

4. Ký hiệu, chữ viết tắt và ký hiệu toán học

FIPS	Federal Information Processing Standards (Tiêu chuẩn Xử lý Thông tin Liên bang)
API	Application Programming Interface (Giao diện lập trình ứng dụng)
PQC	Post-Quantum Cryptography (Mật mã hậu lượng tử)
SHA	Secure Hash Algorithm (Thuật toán hàm băm an toàn)
RBG	Random Bit Generator (bộ sinh bit ngẫu nhiên)
SHAKE	Secure Hash Algorithm KECCAK (Thuật toán hàm băm an toàn KECCAK)
NTT	Number Theoretic Transform (Biến đổi lý thuyết số)
ML-DSA	Module-Lattice-Based Digital Signature Algorithm (Thuật toán chữ ký số dựa trên lưới mô-đun)
SUF-CMA	Strongly existentially UnForgeable under Chosen Message Attack (Tính không thể giả mạo tồn tại mạnh dưới tấn công lựa chọn thông điệp)
MLWE	Module Learning With Errors (Bài toán học có lỗi trên mô-đun)
LWE	Learning With Errors (Bài toán học có lỗi)
MSIS	Module Short Integer Solution (Bài toán nghiệm nguyên ngắn trên mô-đun)
SIS	Short Integer Solution (Bài toán nghiệm nguyên ngắn)
XOF	eXtendable-Output Function (Hàm đầu ra mở rộng)
OID	Object Identifier (Mã định danh đối tượng)
q	Số nguyên tố $q = 2^{23} - 2^{13} + 1 = 8380417$.

$\mathbb{B}$	Tập hợp $\{0, 1, \dots, 255\}$ các số nguyên được biểu diễn bằng một byte.
$\mathbb{N}$	Tập các số tự nhiên $\{1, 2, 3, \dots\}$.
$\mathbb{Z}$	Vành các số nguyên.
$\mathbb{Z}_m$	Vành các số nguyên theo modulo m , có tập phần tử là $\{0, 1, \dots, m - 1\}$.
$\mathbb{Z}_m^n$	Tập các bộ n phần tử trên $\mathbb{Z}_m$ được trang bị cấu trúc $\mathbb{Z}$ -mô-đun.
R	Vành các đa thức một biến trên $\mathbb{Z}$ theo modulo $X^{256} + 1$, còn được ký hiệu là $\mathbb{Z}[X]/(X^{256} + 1)$.
R_m	Vành các đa thức một biến trên $\mathbb{Z}_m$ theo modulo $X^{256} + 1$, còn được ký hiệu là $\mathbb{Z}_m[X]/(X^{256} + 1)$.
B_τ	Tập tất cả các đa thức $p = \sum_{i=0}^{255} p_i X^i$ trong R_q sao cho đúng τ hệ số p_i thuộc tập $\{-1, 1\}$, và tất cả các hệ số còn lại đều bằng 0. (Xem Mục 7.3.)
Π	Được sử dụng để ký hiệu tích trực tiếp của hai hoặc nhiều vành, trong đó phép cộng và phép nhân được thực hiện theo từng thành phần.
T_q	Vành $\Pi_{j=0}^{255} \mathbb{Z}_q$.
$A \times B$	Tích Descartes của hai tập hợp A, B .
$[a, b]$	Với hai số nguyên $a \leq b$, $[a, b]$ ký hiệu tập các số nguyên $\{a, a + 1, \dots, b\}$.
bitlen a	Độ dài bit của số nguyên dương a . Độ dài bit của a là số chữ số xuất hiện trong biểu diễn cơ số 2 của a , với giả thiết chữ số có trọng số lớn nhất luôn bằng 1 (ví dụ, bitlen $32 = 6$ và bitlen $31 = 5$)
BitRev ₈ (r)	Phép đảo ngược bit của số nguyên 8-bit r . Nếu $r = r_0 + 2r_1 + 4r_2 + \dots + 128r_7$ với $r_i \in \{0, 1\}$, thì BitRev ₈ (r) = $r_7 + 2r_6 + 4r_5 + \dots + 128r_0$.
0x	Tiền tố của một số nguyên được biểu diễn ở dạng thập lục phân.
$\log_2 x$	Lôgarit cơ số 2 của x . Ví dụ, $\log_2(16) = 4$.
mod	Nếu α là một số nguyên dương và $m \in \mathbb{Z}$ hoặc $m \in \mathbb{Z}_\alpha$, thì $m \bmod \alpha$ ký hiệu phần tử duy nhất $m' \in \mathbb{Z}$ trong khoảng $0 \leq m' < \alpha$ sao cho m và m' đồng dư theo modulo α .
mod $^\pm$	Nếu α là một số nguyên dương và $m \in \mathbb{Z}$ hoặc $m \in \mathbb{Z}_\alpha$, thì $m \bmod^\pm \alpha$ ký hiệu phần tử duy nhất $m' \in \mathbb{Z}$ trong khoảng $-\lceil \alpha/2 \rceil < m' \leq \lceil \alpha/2 \rceil$ sao cho m và m' đồng dư theo modulo α .
$\lfloor x \rfloor$	Số nguyên lớn nhất không vượt quá số thực x , được gọi là hàm sàn của x . Ví dụ, $\lfloor 2,1 \rfloor = 2$, và $\lfloor 4 \rfloor = 4$.
$\lceil x \rceil$	Số nguyên nhỏ nhất không nhỏ hơn số thực x , được gọi là hàm trần của x . Ví dụ,

$[5] = 5$ và $[5,3] = 6$.

$\|\cdot\|_\infty$ Chuẩn vô cùng. Với phần tử $w \in \mathbb{Z}$, $\|w\|_\infty = |w|$, tức là giá trị tuyệt đối của w . Với phần tử $w \in \mathbb{Z}_q$, $\|w\|_\infty = |w \bmod^\pm q|$. Với phần tử w của R hoặc R_q , $\|w\|_\infty = \max_{0 \leq i < 256} \|w_i\|_\infty$. Với vector $\mathbf{w}$ có độ dài m với các phần tử thuộc R hoặc R_q , $\|\mathbf{w}\|_\infty = \max_{0 \leq i < m} \|w[i]\|_\infty$.

$a!$ Giá trị giai thừa $1 \cdot 2 \cdot 3 \cdot \dots \cdot a$. Giá trị $0!$ được định nghĩa bằng 1.

$\binom{a}{b}$ Với $a \geq b$, là giá trị $a!/(b!(a-b)!)$.

$s \leftarrow x$ Trong giả mã, ký hiệu này có nghĩa là biến s được gán giá trị của biểu thức x .

$s \leftarrow \mathbb{B}^\ell$ Trong giả mã, ký hiệu này có nghĩa là biến s được gán giá trị là một mảng gồm ℓ byte ngẫu nhiên. Các byte này phải được sinh bằng nguồn ngẫu nhiên từ một RBG được quy định trong tiêu chuẩn này.

$x \in S \leftarrow y$ *Ép kiểu*. Biến x được gán một giá trị thuộc tập hợp được xây dựng từ giá trị của biểu thức y thuộc một tập hợp T (có thể khác với S). Tập hợp T và ánh xạ từ T tới S không được quy định tường minh, nhưng phải rõ ràng từ ngữ cảnh xuất hiện của ký hiệu này.

$\llbracket a < b \rrbracket$ *Vị từ Boolean*. Toán tử so sánh nằm trong cặp dấu ngoặc vuông kép $\llbracket a < b \rrbracket$ biểu thị rằng biểu thức phải được đánh giá dưới dạng một giá trị Boolean. Các giá trị Boolean cũng có thể được xem là các phần tử của $\mathbb{Z}_2$, trong đó 1 biểu thị giá trị đúng (true) và 0 biểu thị giá trị sai (false).

$\langle\langle f(x) \rangle\rangle$ Biến tạm dùng để lưu kết quả của phép tính $f(x)$ nhằm cho phép sử dụng lại nhiều lần mà không cần tính toán lại. Điều này tương đương với việc định nghĩa một biến tạm $y \leftarrow f(x)$. Việc đặt tên biến là $\langle\langle f(x) \rangle\rangle$ giúp giả mã trở nên gọn hơn.

$a \parallel b$ Phép nối hai chuỗi bit hoặc chuỗi byte a và b .

$a \circ b$ Phép nhân của a và b trong vành T_q .

$a \cdot b$ hoặc ab Phép nhân trong các vành $\mathbb{Z}$, $\mathbb{Z}_m$, R , hoặc R_m .

$a + b$ Phép cộng của a và b .

a/b Phép chia số nguyên. Khi ký hiệu này được sử dụng, a và b luôn là các số nguyên. Nếu không thể giả định rằng b chia hết a , thì sẽ sử dụng $\lfloor a/b \rfloor$ hoặc $\lceil a/b \rceil$.

$\perp$ Ký hiệu rỗng biểu thị sự thất bại hoặc việc thuật toán không tạo ra đầu ra.

4.1. Quy ước ký hiệu

4.1.1. Vành

Các phần tử của các vành $\mathbb{Z}$, $\mathbb{Z}_q$, $\mathbb{Z}_2$, R , và R_q được ký hiệu bằng các chữ cái thường in nghiêng (ví dụ: w). Các phần tử của vành T_q là các mảng có độ dài 256 gồm các phần tử của $\mathbb{Z}_q$, và được ký hiệu bằng các

chữ cái in nghiêng có dấu mũ (ví dụ: $\hat{w}$). Phép cộng và phép nhân các phần tử của T_q được thực hiện theo từng thành phần. Do đó, thành phần thứ i của tích hai phần tử $\hat{u}$ và $\hat{v}$ thuộc T_q là $\hat{u}[i] \cdot \hat{v}[i] \in \mathbb{Z}_q$. Phép toán nhân trong T_q được ký hiệu bằng $\circ$ (xem mục 4).

Khi một tích $a \cdot b$ hoặc một tổng $a + b$ được viết mà a hoặc b là một lớp đồng dư theo modulo m (tức là, nếu a hoặc b là một phần tử của $\mathbb{Z}_m$ hoặc R_m), thì tích hoặc tổng đó cũng được hiểu là một lớp đồng dư theo modulo m (tức là, một phần tử của $\mathbb{Z}_m$ hoặc R_m). Tương tự, một phần tử của R hoặc $\mathbb{Z}$ có thể được “ép kiểu” thành một phần tử của R_m hoặc $\mathbb{Z}_m$ tương ứng, và có thể được sử dụng làm đầu vào của một hàm được chỉ định để tác động trên một phần tử của R_m hoặc $\mathbb{Z}_m$ tương ứng. Trong cả hai trường hợp, chính phần tử đó hoặc các hệ số của nó được ánh xạ từ $\mathbb{Z}$ sang $\mathbb{Z}_m$ bằng cách lấy lớp đồng dư duy nhất theo modulo m có chứa số nguyên đó.

Các hệ số của một phần tử w thuộc R hoặc R_m được ký hiệu là w_i sao cho $w = w_0 + w_1X + \dots + w_{255}X^{255}$. Nếu w thuộc R (tương ứng, R_m) và t thuộc $\mathbb{Z}$ (tương ứng, $\mathbb{Z}_m$), thì $w(t)$ ký hiệu giá trị của đa thức $w = w_0 + w_1X + \dots + w_{255}X^{255}$ tại giá trị $X = t$.

4.1.2. Vector và ma trận

Các vector có các phần tử thuộc R hoặc R_m được ký hiệu bằng các chữ cái thường in đậm (ví dụ: $\mathbf{v}$). Các ma trận có các phần tử thuộc R hoặc R_m được ký hiệu bằng các chữ cái hoa in đậm (ví dụ: $\mathbf{A}$).

Nếu S là một vành và $\mathbf{v}$ là một vector độ dài L trên S , thì các phần tử của vector $\mathbf{v}$ được biểu diễn là

$$v[0], v[1], \dots, v[L-1]$$

Các phần tử của một ma trận $\mathbf{A}$ kích thước $K \times L$ trên S được ký hiệu là $\mathbf{A}[i, j]$, trong đó $0 \leq i < K$ và $0 \leq j < L$.

Tập hợp tất cả các vector độ dài L trên S được ký hiệu là S^L . Tập hợp tất cả các ma trận kích thước $K \times L$ trên S được ký hiệu là $S^{K \times L}$. Một vector độ dài L cũng có thể được xem như một ma trận kích thước $L \times 1$.

4.2. Biểu diễn NTT

Phép biến đổi lý thuyết số (NTT) là một đẳng cấu cụ thể giữa các vành R_q và T_q . Gọi $\zeta = 1753 \in \mathbb{Z}_q$, là một căn **nguyên thủy** bậc 512 của đơn vị. Nếu $w \in R_q$, thì

$$\text{NTT}(w) = (w(\zeta_0), w(\zeta_1), \dots, w(\zeta_{255})) \in T_q \quad (2.1)$$

trong đó, $\zeta_i = \zeta^{2^{\text{BitRev}_8(i)+1}} \bmod q$. Xem mục 9.5 để biết thảo luận về việc cài đặt cho NTT và NTT^{-1} .

Lý do sử dụng NTT là phép nhân trong vành T_q nhanh hơn đáng kể. Vì NTT là một đẳng cấu, nên với mọi $a, b \in R_q$, ta có:

$$\text{NTT}(ab) = \text{NTT}(a) \circ \text{NTT}(b) \quad (2.2)$$

Nếu $\mathbf{A}$ là một ma trận với các phần tử thuộc R_q , thì $\text{NTT}(\mathbf{A})$ biểu thị ma trận được tính bằng cách áp dụng NTT cho từng phần tử cho $\mathbf{A}$. Ký hiệu $\circ$ cũng được sử dụng để biểu thị phép nhân ma trận đối với các ma trận có các phần tử thuộc T_q . Do đó, $\text{NTT}(\mathbf{AB}) = \text{NTT}(\mathbf{A}) \circ \text{NTT}(\mathbf{B})$. Các thuật toán tường minh cho đại số tuyến tính trên T_q được trình bày trong mục 9.6.

5. Tổng quan về lược đồ chữ ký số ML-DSA

ML-DSA là một lược đồ chữ ký số dựa trên CRYSTALS-DILITHIUM [6]. Lược đồ này bao gồm ba thuật toán chính: ML-DSA.KeyGen (**Thuật toán 1**), ML-DSA.Sign (**Thuật toán 2**), và ML-DSA.Verify (**Thuật toán 3**). Lược đồ ML-DSA sử dụng cấu trúc Fiat-Shamir với các lần hủy bỏ (Fiat-Shamir With Aborts) [10, 11] và mang nhiều nét tương đồng nhất với các lược đồ được đề xuất trong [12, 13].

Tiêu chuẩn này cũng định nghĩa một lược đồ chữ ký có liên quan chặt chẽ nhưng được phân tách miền, gọi là HashML-DSA, khác với ML-DSA ở chỗ có thêm **một bước tiền băm trước khi sinh chữ ký**. Lược đồ này bao gồm ba thuật toán chính: ML-DSA.KeyGen (**Thuật toán 1**), cũng là thuật toán sinh khóa được sử dụng cho ML-DSA; HashML-DSA.Sign (**Thuật toán 4**); và HashML-DSA.Verify (**Thuật toán 5**).

5.1. Các tính chất an toàn

ML-DSA được thiết kế để đạt tính không thể giả mạo tồn tại mạnh dưới tấn công lựa chọn thông điệp (SUF-CMA). Nghĩa là, ta kỳ vọng rằng ngay cả khi một bên đối kháng có thể khiến bên trung thực sinh chữ ký trên các thông điệp tùy ý, thì bên đối kháng đó vẫn không thể tạo ra bất kỳ chữ ký hợp lệ bổ sung nào dựa trên khóa công khai của người ký, kể cả trên các thông điệp mà người ký đã từng sinh chữ ký trước đó.

Bên cạnh tính không thể giả mạo, ML-DSA còn được thiết kế để thỏa mãn các tính chất an toàn bổ sung được mô tả trong [14].

5.2. Các giả thiết tính toán

Tính an toàn của các lược đồ chữ ký số dựa trên lưới thường liên quan đến bài toán Học có lỗi (LWE) và bài toán Nghiệm nguyên ngắn (SIS). Bài toán LWE [15] là bài toán khôi phục một vector $s \in \mathbb{Z}_q^n$ khi cho trước một tập hợp các phương trình tuyến tính “có nhiễu” ngẫu nhiên¹ mà s thỏa mãn. Bài toán SIS là bài toán tìm một nghiệm khác không $t \in \mathbb{Z}_q^n$ của một hệ phương trình tuyến tính cho trước trên $\mathbb{Z}_q$ có dạng $At = 0$ sao cho $\|t\|_\infty$ là nhỏ. Với các lựa chọn tham số thích hợp, những bài toán này là không thể giải được một cách khả thi về mặt tính toán bằng các kỹ thuật tốt nhất hiện đã biết, kể cả phép khử Gauss.

Khi mô-đun $\mathbb{Z}_q^n$ trong các bài toán LWE và SIS được thay thế bằng một mô-đun trên một vành lớn hơn $\mathbb{Z}_q$ (ví dụ: R_q), thì các bài toán tương ứng tạo thành được gọi là Học có lỗi trên Mô-đun (MLWE) [4] và Nghiệm nguyên ngắn trên Mô-đun (MSIS). Độ an toàn của ML-DSA dựa trên bài toán MLWE trên R_q và một biến thể không chuẩn của bài toán MSIS được gọi là SelfTargetMSIS [16].

5.3. Cấu trúc ML-DSA

ML-DSA là một lược đồ chữ ký kiểu Schnorr với một vài tối ưu hóa. Lược đồ chữ ký Schnorr áp dụng **kỹ thuật heuristic Fiat-Shamir** cho một giao thức tương tác giữa một bên xác minh, người biết g (phần tử sinh của một nhóm mà trong đó bài toán logarit rời rạc được tin là khó giải) và giá trị $y = g^x$, với một bên chứng minh, người biết g và x . Giao thức tương tác, nơi bên chứng minh thể hiện sự hiểu biết về x cho bên xác

¹ Cụ thể, bài toán LWE là giải một hệ phương trình có dạng $As + e = b$, trong đó A và b được cho trước, còn e không được cho trước nhưng được biết là nhỏ.

minh, bao gồm ba bước:

1. **Cam kết:** Bên chứng minh sinh một số nguyên dương ngẫu nhiên r nhỏ hơn cấp của g và cam kết giá trị này bằng cách gửi g^r cho bên xác minh.
2. **Thử thách:** Bên xác minh gửi một số nguyên dương ngẫu nhiên c nhỏ hơn cấp của g cho bên chứng minh.
3. **Phản hồi:** Bên chứng minh trả về $s = r - cx$ được rút gọn theo modulo cấp của g , và bên xác minh kiểm tra xem liệu $g^s \cdot y^c = g^r$ có đúng hay không.

Giao thức này được chuyển thành giao thức không tương tác và trở thành một lược đồ chữ ký số bằng cách thay thế việc chọn ngẫu nhiên c của bên xác minh ở bước 2 bằng một quá trình tắt định, trong đó c được dẫn xuất giả ngẫu nhiên từ giá trị băm của cam kết g^r ghép nối với thông điệp cần ký. Đối với lược đồ chữ ký này, y là khóa công khai và x là khóa bí mật.

Ý tưởng cơ bản của ML-DSA và các lược đồ chữ ký số dựa trên lưới tương tự là xây dựng một lược đồ chữ ký số từ một giao thức tương tác tương tự, trong đó một bên chứng minh, người biết các ma trận $\mathbf{A} \in \mathbb{Z}_q^{K \times L}$, $\mathbf{S}_1 \in \mathbb{Z}_q^{L \times n}$, và $\mathbf{S}_2 \in \mathbb{Z}_q^{K \times n}$ với các hệ số nhỏ (đối với $\mathbf{S}_1$ và $\mathbf{S}_2$), chứng minh sự hiểu biết về các ma trận này với một bên xác minh, người biết $\mathbf{A}$ và $\mathbf{T} \in \mathbb{Z}_q^{K \times n} = \mathbf{A}\mathbf{S}_1 + \mathbf{S}_2$. Một giao thức tương tác như vậy sẽ diễn ra như sau:

1. **Cam kết:** Bên chứng minh sinh $\mathbf{y} \in \mathbb{Z}_q^L$ với các hệ số nhỏ và cam kết giá trị này bằng cách gửi $\mathbf{w}_{\text{Approx}} = \mathbf{A}\mathbf{y} + \mathbf{y}_2$ cho bên xác minh, trong đó $\mathbf{y}_2 \in \mathbb{Z}_q^K$ là một vector có các hệ số nhỏ.
2. **Thử thách:** Bên xác minh gửi một vector $\mathbf{c} \in \mathbb{Z}_q^n$ có các hệ số nhỏ cho bên chứng minh.
3. **Phản hồi:** Bên chứng minh trả về $\mathbf{z} = \mathbf{y} + \mathbf{S}_1\mathbf{c}$, và bên xác minh kiểm tra rằng $\mathbf{z}$ có các hệ số nhỏ và $\mathbf{A}\mathbf{z} - \mathbf{T}\mathbf{c} \approx \mathbf{w}_{\text{Approx}}$.

Ở dạng được trình bày, giao thức trên tồn tại một điểm yếu về an toàn: phản hồi $\mathbf{z}$ sẽ bị chệch theo một hướng liên quan đến giá trị bí mật $\mathbf{S}_1$. Tương tự như vậy, $\mathbf{r} = \mathbf{w}_{\text{Approx}} - \mathbf{A}\mathbf{z} + \mathbf{T}\mathbf{c} = \mathbf{y}_2 + \mathbf{S}_2\mathbf{c}$ bị chệch theo một hướng liên quan đến giá trị bí mật $\mathbf{S}_2$. Tuy nhiên, điểm yếu này có thể được khắc phục khi chuyển giao thức tương tác thành một lược đồ chữ ký số. Giống như với chữ ký số Schnorr, người ký sinh giá trị thử thách bằng một quá trình giả ngẫu nhiên từ giá trị băm của cam kết ghép nối với thông điệp. Để khắc phục sự chệch này, người ký áp dụng kỹ thuật lấy mẫu loại bỏ đối với $\mathbf{z}$; nếu các hệ số của $\mathbf{z}$ nằm ngoài khoảng được quy định, thì quá trình sinh chữ ký sẽ bị hủy bỏ và người ký bắt đầu lại với một giá trị $\mathbf{y}$ mới. Tương tự, kỹ thuật lấy mẫu loại bỏ cũng phải được áp dụng cho $\mathbf{r}$. Các phép kiểm tra này tương tự như phép kiểm tra tại Dòng 23 của **Thuật toán 7**. Trong lược đồ chữ ký Fiat-Shamir với các lần hủy bỏ ở dạng đơn giản hóa, khóa công khai là $(\mathbf{A}, \mathbf{T})$ và khóa bí mật là $(\mathbf{S}_1, \mathbf{S}_2)$.

Trong tiêu chuẩn ML-DSA, một số tinh chỉnh và sửa đổi được thêm vào khung cơ sở này vì lý do an toàn hoặc hiệu năng:

- Để giảm kích thước khóa và chữ ký cũng như để sử dụng phép nhân đa thức dựa trên NTT nhanh, ML-DSA sử dụng các ma trận có cấu trúc mô-đun. So với lược đồ cơ sở được mô tả ở trên, ML-DSA thay thế các khối ma trận kích thước $n \times n$ và các khối vector kích thước n bằng các đa thức trong vành R_q . Do đó, thay vì sử dụng $\mathbf{A} \in \mathbb{Z}_q^{K \times L}$, $\mathbf{T} \in \mathbb{Z}_q^{K \times n}$, $\mathbf{S}_1 \in \mathbb{Z}_q^{L \times n}$, $\mathbf{S}_2 \in \mathbb{Z}_q^{K \times n}$, $\mathbf{y} \in \mathbb{Z}_q^L$, $\mathbf{c} \in \mathbb{Z}_q^n$, ML-DSA sử dụng $\mathbf{A} \in R_q^{k \times \ell}$, $\mathbf{t} \in R_q^k$, $\mathbf{s}_1 \in R_q^\ell$, $\mathbf{s}_2 \in R_q^k$, $\mathbf{y} \in R_q^\ell$, $\mathbf{c} \in R_q$, trong đó $\ell = L/n$ và $k = K/n$.
- Để giảm thêm kích thước của khóa công khai, ma trận $\mathbf{A}$ được sinh giả ngẫu nhiên từ một mầm công khai 256-bit ρ , mầm này được đưa vào khóa công khai của ML-DSA thay cho ma trận $\mathbf{A}$.
- Để tiếp tục giảm thêm kích thước khóa công khai, khóa công khai của ML-DSA sử dụng giá trị nén $\mathbf{t}_1$ thay cho $\mathbf{t}$, giá trị này loại bỏ d bit bậc thấp của mỗi hệ số.
- Để đạt được các tính chất vượt ngoài tính không thể giả mạo (BUFF) được nêu trong [14], ML-DSA không trực tiếp ký thông điệp M mà thay vào đó ký một đại diện thông điệp μ , giá trị này được tạo bằng cách băm chuỗi nối của giá trị băm của khóa công khai và M .
- Để giảm kích thước chữ ký, thay vì đưa giá trị cam kết $\mathbf{w}_{\text{Approx}} = \mathbf{A}\mathbf{y} + \mathbf{y}_2$ vào chữ ký, chữ ký ML-DSA sử dụng một phiên bản làm tròn của $\mathbf{w} = \mathbf{A}\mathbf{y}$ làm giá trị cam kết $\mathbf{w}_1$ và chỉ đưa vào giá trị băm $\tilde{c}$ của chuỗi nối $\mu \parallel \mathbf{w}_1$.
- Để đảm bảo bên xác minh có thể tái tạo lại $\mathbf{w}_1$ từ $\mathbf{z}$ và giá trị nén $\mathbf{t}_1$, chữ ký cũng phải chứa một gợi ý $\mathbf{h} \in R_2^k$ được người ký tính toán bằng khóa bí mật của mình.
- Thêm vào đó, để đảm bảo tính đúng đắn, phải bổ sung một giai đoạn lấy mẫu loại bỏ thứ hai (Dòng 28 của **Thuật toán 7**).

Trong tiêu chuẩn này, các từ viết tắt ML-DSA-44, ML-DSA-65, và ML-DSA-87 được sử dụng để tham chiếu đến ML-DSA với các lựa chọn tham số được trình bày trong Bảng 1. Trong các từ viết tắt này, hậu tố bằng số đề cập đến kích thước của ma trận $\mathbf{A}$. Ví dụ, trong ML-DSA-65, ma trận $\mathbf{A}$ là một ma trận kích thước 6×5 trên R_q .

5.4. Ký có dự phòng và Ký tắt định

Để ML-DSA được an toàn, giá trị cam kết $\mathbf{y}$ của người ký không được sử dụng để ký nhiều hơn một thông điệp, và nó không được để kẻ tấn công dễ dàng đoán ra. Điều này đòi hỏi tính ngẫu nhiên. Về nguyên tắc, các giá trị ngẫu nhiên dùng để tạo $\mathbf{y}$ có thể được sinh bằng cách sử dụng nguồn ngẫu nhiên mới tại thời điểm ký hoặc được sinh giả ngẫu nhiên từ thông điệp đang được ký và một giá trị ngẫu nhiên được tính trước, nằm trong khóa bí mật của người ký.

Theo mặc định, tiêu chuẩn này quy định thuật toán ký sử dụng cả hai loại giá trị ngẫu nhiên. Đây được gọi là biến thể "có dự phòng" của quá trình ký. Việc sử dụng nguồn ngẫu nhiên mới trong quá trình ký giúp giảm thiểu các tấn công kênh kề, trong khi việc sử dụng ngẫu nhiên được tính trước giúp bảo vệ hệ thống khỏi khả năng có những sai sót trong bộ tạo số ngẫu nhiên mà người ký sử dụng tại thời điểm ký.

Tiêu chuẩn này cũng cho phép một biến thể hoàn toàn tắt định của quá trình ký trong trường hợp người ký không có quyền truy cập vào nguồn tạo số ngẫu nhiên mới tại thời điểm ký. Tuy nhiên, sự thiếu tính ngẫu

nhien trong biến thể tắt định làm cho rủi ro của các tấn công kênh kề (đặc biệt là các tấn công gây lỗi) trở nên khó giảm thiểu hơn. Do đó, không nên sử dụng biến thể này trên các nền tảng mà tấn công kênh kề là vấn đề đáng quan tâm và không thể được giảm thiểu bằng biện pháp khác.

Chỉ triển khai biến thể có dự phòng (tức là không có biến thể tắt định) là đủ để đảm bảo khả năng liên tác. Cùng một thuật toán xác minh sẽ hoạt động để kiểm tra các chữ ký được tạo ra bởi một trong hai biến thể, do đó việc cài đặt thêm biến thể tắt định bên cạnh biến thể có dự phòng không nâng cao khả năng liên tác. Quá trình ký có dự phòng và ký tắt định chỉ khác nhau ở Dòng 5 của **Thuật toán 2** và Dòng 5 của **Thuật toán 4**.

5.5. Sử dụng chữ ký số

Quản lý khóa an toàn là một yêu cầu thiết yếu đối với việc sử dụng chữ ký số. Điều này phụ thuộc vào từng ngữ cảnh và bao gồm nhiều nội dung hơn các thuật toán tạo khóa, ký và xác minh chữ ký được nêu trong tiêu chuẩn này. Hướng dẫn quản lý khóa được cung cấp trong bộ tài liệu SP 800-57 [9,17,18].

Chữ ký số hữu ích nhất khi được gắn với một danh tính. Việc gắn một khóa công khai với một danh tính đòi hỏi phải có bằng chứng về việc sở hữu khóa bí mật. Trong ngữ cảnh PKI, việc cấp phát chứng thư số bao gồm các đảm bảo về danh tính và bằng chứng sở hữu. Khi không có chứng thư khóa công khai, người sử dụng chữ ký số nên xác định xem một khóa công khai có cần được gắn với một danh tính hay không. Các phương pháp để đạt được sự đảm bảo về danh tính và bằng chứng sở hữu được trình bày trong [3].

5.6. Các yêu cầu bổ sung

Mục này mô tả một số bảo đảm bắt buộc khi triển khai ML-DSA. Đây là những yêu cầu bổ sung ngoài các vấn đề cần cân nhắc trong Mục 5.5.

5.6.1. Sinh các giá trị ngẫu nhiên

Thuật toán 1, thực hiện việc sinh khóa cho ML-DSA, sử dụng một RBG để sinh ra một mầm (seed) ngẫu nhiên 256-bit ξ . Mầm ξ phải là một giá trị ngẫu nhiên mới (nghĩa là, chưa từng được sử dụng trước đó) được sinh bằng cách sử dụng một RBG phù hợp TCVN 12853:2020 (hoặc được quy định trong SP 800-90A, SP 800-90B và SP 800-90C [19, 20, 21]). Hơn nữa, RBG được sử dụng phải có độ mạnh an toàn ít nhất là 192 bit đối với ML-DSA-65 và 256 bit đối với ML-DSA-87. Đối với ML-DSA-44, RBG nên có độ mạnh an toàn ít nhất là 192 bit và phải có độ mạnh an toàn ít nhất là 128 bit. Nếu sử dụng một RBG với độ mạnh an toàn ít nhất 128 bit nhưng dưới 192 bit, thì độ mạnh an toàn được công bố của ML-DSA-44 sẽ bị giảm từ cấp độ 2 xuống cấp độ 1.

Thêm vào đó, giá trị rnd được sinh ra bằng cách sử dụng một RBG trong các biến thể "có dự phòng" mặc định của **Thuật toán 2** và **Thuật toán 4** tương ứng với việc ký ML-DSA và HashML-DSA. Mặc dù lý tưởng nhất là giá trị này nên được sinh bằng một RBG được quy định trong tiêu chuẩn này, nhưng vẫn có thể sử dụng các phương pháp khác để sinh các giá trị ngẫu nhiên mới. Mục đích chính của rnd là hỗ trợ các biện pháp đối phó với các tấn công kênh kề và các tấn công gây lỗi trên các chữ ký tắt định, chẳng hạn như trong

[22, 23, 24]². Vì mục đích này, ngay cả việc dùng một RBG yếu cũng có thể được ưu tiên hơn so với các biến thể hoàn toàn bất định của **Thuật toán 2** và **Thuật toán 4**.

5.6.2. Kiểm tra độ dài khóa công khai và chữ ký

Thuật toán 3, cài đặt việc xác minh cho ML-DSA, và **Thuật toán 5**, cài đặt việc xác minh cho HashML-DSA, quy định độ dài của chữ ký σ và khóa công khai pk theo các tham số được mô tả trong Bảng 1. Nếu một bản cài đặt của ML-DSA có thể chấp nhận các đầu vào σ hoặc pk có bất kỳ độ dài nào khác, thì bản cài đặt đó phải trả về false bất cứ khi nào độ dài của một trong hai đầu vào này khác với độ dài được quy định trong tiêu chuẩn này. Việc không kiểm tra độ dài của pk hoặc σ có thể làm ảnh hưởng đến các tính chất an toàn mà ML-DSA được thiết kế để đạt được, chẳng hạn như tính không thể giả mạo mạnh.

5.6.3. Các giá trị trung gian

Dữ liệu được sử dụng nội bộ bởi các thuật toán sinh khóa và thuật toán ký trong các bước tính toán trung gian có thể bị kẻ tấn công lợi dụng để thu được thông tin về khóa bí mật và từ đó làm suy yếu tính an toàn. Dữ liệu được sử dụng nội bộ bởi các thuật toán xác minh cũng nhạy cảm tương tự đối với một số ứng dụng, bao gồm việc xác minh các chữ ký được sử dụng làm mã thông báo xác thực theo quyền sở hữu (nghĩa là các bí mật xác thực) hoặc xác minh các chữ ký trên các thông điệp bản rõ được yêu cầu giữ bí mật. Các giá trị trung gian của thuật toán xác minh có thể tiết lộ thông tin về các đầu vào của nó (nghĩa là thông điệp, chữ ký và khóa công khai), và trong một số ứng dụng, yêu cầu về tính an toàn hoặc quyền riêng tư đòi hỏi một hoặc nhiều đầu vào này phải được giữ bí mật. Do đó, các bản cài đặt của ML-DSA phải đảm bảo rằng bất kỳ dữ liệu trung gian nào có khả năng nhạy cảm đều được tiêu hủy ngay khi không còn cần thiết.

Có hai trường hợp cụ thể mà các bản cài đặt có thể không cần phải tiêu hủy dữ liệu trung gian là:

1. Mầm ξ được sinh trong bước 1 của ML-DSA.KeyGen có thể được lưu trữ cho mục đích mở rộng sau này sử dụng ML-DSA.KeyGen_internal. Vì mầm có thể được sử dụng để tính toán khóa bí mật, nó là dữ liệu nhạy cảm và phải được xử lý với các biện pháp bảo vệ tương tự như khóa bí mật.
2. Ma trận $\hat{A}$ được sinh trong bước 3 của ML-DSA.KeyGen_internal có thể được lưu trữ để không cần phải tính lại trong các thao tác sau này. Tương tự như vậy, ma trận $\hat{A}$ được sinh trong bước 5 của thuật toán xác minh ML-DSA.Verify_internal cũng có thể được lưu trữ. Trong cả hai trường hợp, ma trận $\hat{A}$ là dữ liệu dễ dàng tính toán được từ khóa công khai và không yêu cầu bất kỳ biện pháp bảo vệ đặc biệt nào.

Trong một số tình huống nhất định (ví dụ: ký tắt định, xác minh các thông điệp và chữ ký bí mật), phải thận trọng hơn để bảo vệ các bản cài đặt trước các tấn công kênh kẻ hoặc tấn công gây lỗi. Một thiết bị mật mã có thể làm rò rỉ thông tin quan trọng thông qua các kênh kẻ, cho phép trích xuất dữ liệu nội bộ hoặc dữ liệu khóa mà không cần phá vỡ các nguyên thủy mật mã.

² Ngoài ra, khi việc ký là bất định, thông tin về thông điệp bị rò rỉ qua các kênh kẻ thời gian, nhưng thông tin về khóa bí mật thì không. Nếu người ký không muốn tiết lộ thông điệp đang được ký, nên sử dụng chữ ký có dự phòng (xem Mục 5.2 trong [6]).

5.6.4. Không sử dụng số học dấu phẩy động

Các bản cài đặt của ML-DSA **không được** sử dụng số học dấu phẩy động, vì sai số làm tròn trong các phép toán dấu phẩy động có thể dẫn đến kết quả sai trong một số trường hợp. $\lfloor x/y \rfloor$ hoặc $\lceil x/y \rceil$ được sử dụng trong tất cả các giả mã của tiêu chuẩn này ở những nơi thực hiện phép chia (ví dụ: x/y), và y có thể không là ước của x . Cả hai giá trị này đều có thể được tính toán mà không cần số học dấu phẩy động, vì phép chia số nguyên thông thường x/y sẽ tính ra $\lfloor x/y \rfloor$, và $\lceil x/y \rceil = \lfloor (x + y - 1)/y \rfloor$ đối với số nguyên không âm x và số nguyên dương y . Nếu y là một lũy thừa của hai, việc sử dụng các phép toán dịch bit có thể hiệu quả hơn so với phép chia số nguyên.

5.7. Sử dụng mật mã đối xứng

Tiêu chuẩn này sử dụng các hàm SHAKE256 và SHAKE128, như được định nghĩa trong FIPS 202 [7]. Mặc dù FIPS 202 quy định các hàm này nhận đầu vào và xuất đầu ra là các chuỗi bit, hầu hết các bản cài đặt đều coi đầu vào và đầu ra là các chuỗi byte.

Tiêu chuẩn này sẽ luôn gọi các hàm này với độ dài đầu ra là một bội số của 8 bit và xử lý đầu ra của các hàm này như một chuỗi byte. Chuỗi byte này cũng chính là kết quả thu được nếu lấy chuỗi bit kỳ vọng từ FIPS 202 và xử lý nó bằng hàm BitsToBytes. Tuy nhiên, để cho phép ký các thông điệp không có số lượng tròn byte (là bội số của 8 bit), tiêu chuẩn này sẽ nạp chồng hàm SHAKE256 sao cho đầu vào của nó có thể là một chuỗi bit nhưng thông thường sẽ là một chuỗi byte. Đăng thức dưới đây sẽ luôn đúng với bất kỳ chuỗi byte str nào và số nguyên $\ell \geq 1$:

$$\text{SHAKE256}(str, 8\ell) = \text{SHAKE256}(\text{BytesToBits}(str), 8\ell).$$

Ngoài việc sử dụng một biến thể chủ yếu hướng byte của API được định nghĩa trong FIPS 202 cho SHAKE256 và SHAKE128, tiêu chuẩn này đôi khi sử dụng API gia tăng được định nghĩa trong SP 800-185 [25]. API này gồm ba hàm cho mỗi biến thể SHAKE. Các hàm này có thể được sử dụng để hấp thụ một dãy chuỗi có độ dài tùy ý và trích xuất một dãy chuỗi có độ dài tùy ý. Các hàm thực hiện đệm để xử lý mọi khối dữ liệu chưa đầy trong quá trình hấp thụ hoặc trích xuất. Ví dụ, đối với SHAKE256:

- $ctx \leftarrow \text{SHAKE256.Init}()$
Khởi tạo một ngữ cảnh hàm băm.
- $ctx \leftarrow \text{SHAKE256.Absorb}(ctx, str)$
Đưa dữ liệu vào để sử dụng trong giai đoạn hấp thụ của SHAKE256 và cập nhật ngữ cảnh ctx .
- $(ctx, out) \leftarrow \text{SHAKE256.Squeeze}(ctx, 8\ell)$
Trích xuất ℓ byte đầu ra được tạo trong giai đoạn trích xuất của SHAKE256 và cập nhật ngữ cảnh ctx .

Mặc dù các hàm trên được đặc tả theo phép hoán vị Keccak- f thay vì theo hàm đầu ra mở rộng (XOF) SHAKE256, chúng được định nghĩa sao cho mọi dãy lệnh có dạng sau:

1. $\text{ctx} \leftarrow \text{SHAKE256.Init}()$
2. **For** $i = 1$ to m : $\text{ctx} \leftarrow \text{SHAKE256.Absorb}(\text{ctx}, \text{str}_i)$
3. **For** $j = 1$ to k : $(\text{ctx}, \text{out}_j) \leftarrow \text{SHAKE256.Squeeze}(\text{ctx}, 8b_j)$
4. $\text{output} \leftarrow \text{out}_1 \parallel \dots \parallel \text{out}_k$

đều cho cùng đầu ra như một lần gọi SHAKE256 duy nhất:

$$\text{output} \leftarrow \text{SHAKE256}(\text{str}_1 \parallel \dots \parallel \text{str}_m, 8b_1 + \dots + 8b_k).$$

Sự tương đương này đúng bất kể $|\text{str}_i|$ và b_j có phải là bội số của độ dài khối SHAKE256 hay không

Vì tất cả các đầu ra của SHAKE128 và SHAKE256 trong tiêu chuẩn này đều cho kết quả có độ dài tròn byte, các hàm bao H và G được định nghĩa như sau:

1. $H(\text{str}, \ell) = \text{SHAKE256}(\text{str}, 8\ell)$
2. $G(\text{str}, \ell) = \text{SHAKE128}(\text{str}, 8\ell)$
3. $H.\text{Init}() = \text{SHAKE256.Init}()$
4. $G.\text{Init}() = \text{SHAKE128.Init}()$
5. $H.\text{Absorb}(\text{ctx}, \text{str}) = \text{SHAKE256.Absorb}(\text{ctx}, \text{str})$
6. $G.\text{Absorb}(\text{ctx}, \text{str}) = \text{SHAKE128.Absorb}(\text{ctx}, \text{str})$
7. $H.\text{Squeeze}(\text{ctx}, \ell) = \text{SHAKE256.Squeeze}(\text{ctx}, 8\ell)$
8. $G.\text{Squeeze}(\text{ctx}, \ell) = \text{SHAKE128.Squeeze}(\text{ctx}, 8\ell)$

Ngoài SHAKE128 và SHAKE256, HashML-DSA.Sign và HashML-DSA.Verify có thể gọi các hàm băm khác được quy định trong tiêu chuẩn này cho quá trình băm trước. Giả mã trong tiêu chuẩn này cũng coi các hàm này trả về một chuỗi byte làm đầu ra trong khi vẫn hỗ trợ đầu vào là một chuỗi bit hoặc một chuỗi byte. Ở đây, cần lưu ý rằng các hàm băm được định nghĩa trong [8] sử dụng các quy tắc khác nhau (nghĩa là, thứ tự big-endian) để liên kết các bit, byte, và các từ.

6. Các bộ tham số

Ba bộ tham số ML-DSA được trình bày trong Bảng 1. Mỗi bộ tham số gán các giá trị cho tất cả các tham số được sử dụng trong các thuật toán ML-DSA để sinh khóa, sinh chữ ký và xác minh chữ ký. Nhằm cung cấp thông tin, một số tham số được sử dụng trong quá trình phân tích các thuật toán này cũng được đưa vào bảng. Cụ thể, “số lần lặp” chỉ số lần lặp kỳ vọng của vòng lặp chính trong thuật toán ký, theo Công thức 5 trong [5]. Tên của các bộ tham số có dạng “ML-DSA- $k\ell$ ”, trong đó (k, ℓ) là kích thước của ma trận A.

Bảng 1 Các bộ tham số của ML-DSA

Tham số (xem Mục 8.1 và 8.2 của tiêu chuẩn này)	Các giá trị được gán bởi mỗi bộ tham số		
	ML-DSA-44	ML-DSA-65	ML-DSA-87
q - modulo [xem 8.1]	8380417	8380417	8380417
ζ – căn nguyên thủy bậc 512 của đơn vị trong $\mathbb{Z}_q$ [xem 9.5]	1753	1753	1753
d - số lượng bit bị loại bỏ khỏi t [xem 8.1]	13	13	13
τ - số lượng hệ số bằng ± 1 trong đa thức c [xem 8.2]	39	49	60
λ - độ mạnh kháng va chạm của $\tilde{c}$ [xem 8.2]	128	192	256
γ_1 - khoảng giá trị các hệ số của y [xem 8.2]	2^{17}	2^{19}	2^{19}
γ_2 - khoảng làm tròn phần bậc thấp [xem 8.2]	$(q - 1)/88$	$(q - 1)/32$	$(q - 1)/32$
(k, ℓ) - kích thước của A [xem 8.1]	(4,4)	(6,5)	(8,7)
η – khoảng giá trị khóa bí mật [xem 8.1]	2	4	2
$\beta = \tau \cdot \eta$ [xem 8.2]	78	196	120
ω - số bit 1 tối đa trong gọi ý h [xem 8.2]	80	55	75
Entropy của thử thách $\log_2 \binom{256}{\tau} + \tau$	192	225	257
Số lần lặp kỳ vọng	4.36	5.14	3.91
Độ mạnh an toàn được công bố	Cấp độ 2	Cấp độ 3	Cấp độ 5

Các bộ tham số này được thiết kế để đáp ứng những cấp an toàn nhất định do NIST định nghĩa trong Thông báo mời đề xuất ban đầu [26]. Các cấp an toàn này được giải thích thêm trong SP 800-57, Phần 1 [9].

Theo cách tiếp cận này, độ mạnh an toàn không được mô tả bằng một con số duy nhất, chẳng hạn như “128 bit an toàn”. Thay vào đó, mỗi bộ tham số ML-DSA được công bố là an toàn ít nhất ngang với một mã khối tổng quát có kích thước khóa được quy định hoặc một hàm băm tổng quát có độ dài đầu ra được quy định. Chính xác hơn, tuyên bố này khẳng định rằng tài nguyên tính toán cần thiết để phá ML-DSA lớn hơn hoặc bằng tài nguyên tính toán cần thiết để phá mã khối hoặc hàm băm khi các tài nguyên này được ước lượng bằng bất kỳ mô hình tính toán thực tế nào. Các mô hình tính toán khác nhau có thể sát thực tế hơn hoặc kém hơn và, theo đó, dẫn đến các ước lượng độ mạnh an toàn chính xác hơn hoặc kém hơn. Một số mô hình thường được nghiên cứu được thảo luận trong [27].

Cụ thể, bộ tham số ML-DSA-44 được công bố thuộc độ mạnh an toàn cấp độ 2, ML-DSA-65 thuộc độ mạnh an toàn cấp độ 3 và ML-DSA-87 thuộc độ mạnh an toàn cấp độ 5 [6]. Xem [28] để biết thêm thảo luận về độ mạnh an toàn của các hệ mật dựa trên MLWE.

Kích thước của các khóa và chữ ký tương ứng với mỗi bộ tham số được trình bày trong Bảng 2. Có thể thực hiện một số tối ưu hóa nhất định khi lưu trữ khóa công khai và khóa bí mật của ML-DSA. Nếu có thêm dung lượng lưu trữ, thì có thể tính toán trước và lưu trữ $\hat{A}$ để tăng tốc độ ký và xác minh. Mặt khác, nếu muốn giảm thiểu không gian cần thiết cho khóa bí mật, thì có thể chỉ cần lưu trữ m 32 byte, giá trị này là đủ để tạo ra các phần còn lại của khóa bí mật. Để biết thêm chi tiết, xem Mục 5.1 trong [6].

Bảng 2: Kích thước (tính theo byte) của khóa và chữ ký của ML-DSA

	Khóa bí mật	Khóa công khai	Kích thước chữ ký
ML-DSA-44	2560	1312	2420
ML-DSA-65	4032	1952	3309
ML-DSA-87	4896	2592	4627

7. Các hàm bên ngoài

Các hàm ký, xác minh và tạo khóa có thể được chia thành các thành phần “bên ngoài” và “bên trong” để đơn giản hóa các API và quá trình kiểm thử của chương trình xác nhận tính đúng đắn các thuật toán mật mã. Các thành phần bên ngoài tạo ra tính ngẫu nhiên và thực hiện các bước kiểm tra khác nhau trước khi gọi các thành phần bên trong tương ứng của chúng. Các thành phần bên trong mang tính tất định và có thể giả định rằng các thành phần bên ngoài đã không gặp phải các tình trạng lỗi.

Sự phân biệt giữa các hàm bên ngoài và bên trong cũng làm đơn giản hóa việc trình bày các thuật toán ký và xác minh bằng cách gộp các thao tác được chia sẻ giữa ML-DSA.Sign và HashML-DSA.Sign vào ML-DSA.Sign_internal và gộp các thao tác được chia sẻ giữa ML-DSA.Verify và HashML-DSA.Verify vào ML-DSA.Verify_internal.

7.1. Tạo khóa ML-DSA

Thuật toán tạo khóa ML-DSA.KeyGen không nhận đầu vào, đưa ra một khóa công khai và một khóa bí mật, cả hai đều được mã hóa dưới dạng chuỗi byte.

Thuật toán sử dụng một RBG được quy định trong tiêu chuẩn này để tạo ra một mầm ξ ngẫu nhiên 256-bit (32 byte), giá trị này được cung cấp làm đầu vào cho ML-DSA.KeyGen_internal (**Thuật toán 6**), thuật toán này sẽ tạo ra khóa công khai và khóa bí mật.

Thuật toán 1 ML-DSA.KeyGen () – <i>Sinh một cặp khóa công khai-bí mật</i>	
Đầu ra: Khóa công khai $pk \in \mathbb{B}^{32+32k(\text{bitlen}(q-1)-d)}$ và khóa bí mật $sk \in \mathbb{B}^{32+32+64+32\cdot((\ell+k)\cdot \text{bitlen}(2\eta)+dk)}$	
1: $\xi \leftarrow \mathbb{B}^{32}$	▷ chọn mầm ngẫu nhiên
2: if $\xi = \perp$ then	
3: return $\perp$	▷ trả về chỉ báo lỗi nếu quá trình sinh bit ngẫu nhiên thất bại
4: end if	
5: return ML-DSA.KeyGen_internal (ξ)	

7.2. Ký số ML-DSA

Thuật toán ký ML-DSA.Sign (**Thuật toán 2**) nhận một khóa bí mật, một thông điệp, và một chuỗi ngữ cảnh làm đầu vào. Nó đưa ra một chữ ký số được mã hóa dưới dạng chuỗi byte.

Đối với phiên bản “có dự phòng” mặc định của quá trình ký số ML-DSA, thuật toán (tại dòng 5) sử dụng một RBG được quy định trong tiêu chuẩn này để tạo ra một mầm ngẫu nhiên 256-bit (32 byte) rnd . Nếu muốn sử dụng biến thể tắt định, thì rnd được đặt thành chuỗi số 0 cố định $\{0\}^{32}$. Giá trị rnd , khóa bí mật, và thông điệp đã mã hóa được đưa vào làm đầu vào cho ML-DSA.Sign_internal (**Thuật toán 7**), thuật toán này sẽ tạo ra chữ ký.

Thuật toán 2 ML-DSA.Sign(sk, M, ctx) – <i>Sinh chữ ký ML-DSA</i>	
Đầu vào: Khóa bí mật $sk \in \mathbb{B}^{32+32+64+32\cdot((\ell+k)\cdot \text{bitlen}(2\eta)+dk)}$, thông điệp $M \in \{0,1\}^*$, chuỗi ngữ cảnh ctx (chuỗi byte có độ dài 255 byte hoặc ít hơn).	
Đầu ra: Chữ ký $\sigma \in \mathbb{B}^{\lambda/4+\ell\cdot 32\cdot(1+\text{bitlen}(\gamma_1-1))+\omega+k}$.	
1: if $ ctx > 255$ then	
2: return $\perp$	▷ trả về chỉ báo lỗi nếu chuỗi ngữ cảnh quá dài
3: end if	
4:	

5: $rnd \leftarrow \mathbb{B}^{32}$	▷ đối với biến thể tắt định tùy chọn, thay thế bằng $rnd \leftarrow \{0\}^{32}$
6: if $rnd = \perp$ then	
7: return $\perp$	▷ trả về chỉ báo lỗi nếu quá trình sinh bit ngẫu nhiên thất bại
8: end if	
9:	
10: $M' \leftarrow \text{BytesToBits}(\text{IntegerToBytes}(0,1) \parallel \text{IntegerToBytes}(ctx , 1) \parallel ctx) \parallel M$	
11: $\sigma \leftarrow \text{ML} - \text{DSA}.\text{Sign_internal}(sk, M', rnd)$	
12: return σ	

7.3. Xác minh ML-DSA

Thuật toán xác minh ML-DSA.Verify (**Thuật toán 3**) nhận một khóa công khai, một thông điệp, một chữ ký, và một chuỗi ngữ cảnh làm đầu vào. Khóa công khai, chữ ký, và chuỗi ngữ cảnh đều được mã hóa dưới dạng các chuỗi byte, trong khi thông điệp là một chuỗi bit. ML-DSA.Verify đưa ra một giá trị Boolean, mang giá trị đúng (true) nếu chữ ký là hợp lệ đối với thông điệp và khóa công khai, và mang giá trị sai (false) nếu chữ ký không hợp lệ. Việc xác minh được thực hiện bằng cách gọi ML-DSA.Verify_internal (**Thuật toán 8**) với các đầu vào gồm khóa công khai, thông điệp đã được mã hóa và chữ ký.

Thuật toán 3 ML-DSA.Verify(pk, M, σ, ctx) – Xác minh chữ ký σ đối với thông điệp M	
Đầu vào: Khóa công khai $pk \in \mathbb{B}^{32+32k}$ (bitlen $(q-1)-d$), thông điệp $M \in \{0,1\}^*$, chữ ký $\sigma \in \mathbb{B}^{\lambda/4+\ell \cdot 32 \cdot (1+\text{bitlen}(\gamma_1-1))+\omega+k}$, chuỗi ngữ cảnh ctx (một chuỗi byte có độ dài 255 byte hoặc ít hơn).	
Đầu ra: Giá trị Boolean	
1: if $ ctx > 255$ then	
2: return false	▷ trả về chỉ báo lỗi nếu chuỗi ngữ cảnh quá dài
3: end if	
4:	
5: $M' \leftarrow \text{BytesToBits}(\text{IntegerToBytes}(0,1) \parallel \text{IntegerToBytes}(ctx , 1) \parallel ctx) \parallel M$	
6: return ML – DSA.Verify_internal(pk, M', σ)	

7.4. ML-DSA dạng băm trước

Đối với một số mô-đun mật mã tạo chữ ký ML-DSA, việc băm thông điệp ở bước 6 của ML-DSA.Sign_internal có thể dẫn đến hiệu suất không thể chấp nhận được nếu thông điệp M lớn. Ví dụ, nền tảng có thể yêu cầu hỗ trợ phần cứng cho việc băm để đạt được hiệu suất có thể chấp nhận được nhưng lại thiếu hỗ trợ phần cứng dành riêng cho SHAKE256. Đối với một số trường hợp sử dụng, vấn đề này có thể được giải quyết bằng cách ký một bản tóm tắt của thông điệp cùng với một số thông tin phân tách miền thay vì ký trực tiếp lên thông điệp. Phiên bản này của ML-DSA được gọi là ML-DSA băm trước hoặc HashML-DSA.

Nhìn chung, phiên bản ML-DSA thuần túy được ưu tiên sử dụng hơn.

Mặc dù việc tạo khóa cho HashML-DSA giống hệt với ML-DSA, nhưng điều này không đúng đối với thuật toán ký HashML-DSA.Sign hoặc thuật toán xác minh HashML-DSA.Verify. Giống như ML-DSA, thuật toán ký của HashML-DSA nhận nội dung cần ký, khóa bí mật, và một ngữ cảnh làm đầu vào, cũng như một hàm băm hoặc XOF sẽ được sử dụng để băm trước nội dung cần ký. Chuỗi ngữ cảnh có độ dài tối đa là 255 byte. Theo mặc định, ngữ cảnh là chuỗi rỗng, mặc dù các ứng dụng có thể chỉ định sử dụng một chuỗi ngữ cảnh không rỗng.

Định danh cho một chữ ký (ví dụ: định danh đối tượng [OID]) nên chỉ ra liệu chữ ký đó là chữ ký ML-DSA hay chữ ký băm trước HashML-DSA. Trong trường hợp chữ ký băm trước, định danh cũng nên chỉ ra hàm băm hoặc XOF được sử dụng để tính toán giá trị băm trước³. Mặc dù một cặp khóa duy nhất có thể được sử dụng cho cả chữ ký ML-DSA và HashML-DSA, nhưng khuyến nghị rằng mỗi cặp khóa chỉ nên được sử dụng cho phiên bản này hoặc phiên bản kia. Nếu sử dụng một chuỗi ngữ cảnh không rỗng, điều này nên được biểu thị thông qua định danh của chữ ký hoặc thông qua ứng dụng đang sử dụng chữ ký đó.

Nếu biến thể “có dự phòng” mặc định được sử dụng, giá trị ngẫu nhiên 32-byte rnd phải được tạo ra bởi module mật mã tạo chữ ký (nghĩa là module chạy hàm ML-DSA.Sign_internal). Tuy nhiên, tất cả các bước ký khác có thể được thực hiện bên ngoài module mật mã tạo chữ ký. Trong trường hợp băm trước, hàm băm hoặc XOF của nội dung cần ký phải được tính toán bên trong một module mật mã đã được xác thực theo chuẩn FIPS 140 và có thể là một module mật mã khác với module tạo ra chữ ký.

Nếu nội dung cần ký có kích thước lớn, việc băm nội dung thường được thực hiện ở cấp ứng dụng. Ví dụ, trong Cú pháp thông điệp mật mã (CMS) [29], một bản tóm tắt của nội dung có thể được tính toán, và bản tóm tắt đó được ký cùng với các thuộc tính khác. Nếu nội dung không được băm ở cấp ứng dụng, phiên bản băm trước của quá trình ký ML-DSA có thể được sử dụng.

Để duy trì cùng một mức độ mạnh an toàn khi nội dung được băm ở cấp ứng dụng hoặc khi sử dụng HashML-DSA, bản tóm tắt được ký cần phải được tạo ra bằng cách sử dụng một hàm băm hoặc XOF được quy định trong tiêu chuẩn này (ví dụ: từ FIPS 180 [8] hoặc FIPS 202 [7]) cung cấp ít nhất λ bit độ mạnh an toàn cổ điển chống lại cả các tấn công tìm và chạm và tấn công tiền ảnh thứ hai (Xem [7], Bảng 4)⁴. Việc xác minh một chữ ký được tạo ra theo cách này sẽ yêu cầu hàm xác minh phải tạo ra một bản tóm tắt từ thông điệp theo cùng một cách để sử dụng làm đầu vào cho hàm xác minh.

7.4.1. Ký và Xác minh HashML-DSA

Trong phiên bản HashML-DSA, thông điệp đầu vào cho ML-DSA.Sign_internal là kết quả của việc áp dụng một hàm băm hoặc một XOF cho nội dung cần ký. Đầu ra của hàm băm hoặc XOF được thêm vào trước bởi một dấu phân tách miền một byte, một byte cho biết độ dài của chuỗi ngữ cảnh, chuỗi ngữ cảnh và mã hóa DER (quy tắc mã hóa phân biệt) của OID của hàm băm hoặc XOF. Dấu phân tách miền có giá trị là một

³ Trong trường hợp sử dụng một XOF, điều này cũng bao gồm độ dài đầu ra của XOF.

⁴ Để đạt độ mạnh an toàn cổ điển ít nhất λ bit trước các tấn công va chạm, giá trị băm cần ký phải có độ dài ít nhất 2λ bit

đối với việc ký “dạng trước băm”. Mã hóa DER của OID bao gồm thẻ và độ dài.

Thuật toán 4 hiển thị mã hóa DER của các OID cho SHA-256, SHA-512 và SHAKE128. Tuy nhiên, nó có thể được sử dụng với các hàm băm khác hoặc XOF.

Thuật toán 4 HashML – DSA. $\text{Sign}(sk, M, ctx, PH)$ – <i>Sinh một chữ ký ML-DSA dạng băm trước</i>	
Đầu vào: Khóa bí mật $sk \in \mathbb{B}^{32+32+64+32 \cdot ((\ell+k) \cdot \text{bitlen}(2\eta)+dk)}$, thông điệp $M \in \{0,1\}^*$, chuỗi ngữ cảnh ctx (một chuỗi byte có độ dài 255 byte hoặc ít hơn), hàm băm trước PH.	
Đầu ra: Chữ ký ML-DSA $\sigma \in \mathbb{B}^{\lambda/4+\ell \cdot 32 \cdot (1+\text{bitlen}(\gamma_1-1))+\omega+k}$.	
1: if $ ctx > 255$ then	
2: return $\perp$	▷ trả về chỉ báo lỗi nếu chuỗi ngữ cảnh quá dài
3: end if	
4:	
5: $rnd \leftarrow \mathbb{B}^{32}$	▷ đối với biến thể tắt định tùy chọn, thay thế bằng $rnd \leftarrow \{0\}^{32}$
6: if $rnd = \perp$ then	
7: return $\perp$	▷ trả về chỉ báo lỗi nếu quá trình sinh bit ngẫu nhiên thất bại
8: end if	
9:	
10: switch PH do	
11: case SHA – 256:	
12: OID $\leftarrow$ 0x06, 0x09, 0x60, 0x86, 0x48, 0x01, 0x65, 0x03, 0x04, 0x02, 0x01	▷ 2.16.840.1.101.3.4.2.1
13: $PH_M \leftarrow \text{SHA256}(M)$	
14: case SHA – 512:	
15: OID $\leftarrow$ 0x06, 0x09, 0x60, 0x86, 0x48, 0x01, 0x65, 0x03, 0x04, 0x02, 0x03	▷ 2.16.840.1.101.3.4.2.3
16: $PH_M \leftarrow \text{SHA512}(M)$	
17: case SHAKE128:	
18: OID $\leftarrow$ 0x06, 0x09, 0x60, 0x86, 0x48, 0x01, 0x65, 0x03, 0x04, 0x02, 0x0B	▷ 2.16.840.1.101.3.4.2.11
19: $PH_M \leftarrow \text{SHAKE128}(M, 256)$	
20: case ...	
21: ...	
22: end switch	

```

23:  $M' \leftarrow \text{BytesToBits}(\text{IntegerToBytes}(1,1) \parallel \text{IntegerToBytes}(|ctx|, 1) \parallel ctx \parallel \text{OID} \parallel \text{PH}_M)$ 
24:  $\sigma \leftarrow \text{ML-DSA.Sign\_internal}(sk, M', rnd)$ 
25: return  $\sigma$ 

```

Thuật toán 5 trình bày quá trình xác minh chữ ký cho HashML-DSA. Hàm này xây dựng M' theo cùng một cách như **Thuật toán 4** và truyền M' kết quả vào thuật toán ML-DSA.Verify_internal để xác minh. Giống như quá trình tạo chữ ký băm trước, M' có thể được xây dựng bên ngoài module mật mã thực hiện ML-DSA.Verify_internal. Tuy nhiên, trong trường hợp của HashML-DSA, hàm băm hoặc XOF của nội dung phải được tính toán bên trong một module mật mã đã được xác thực theo chuẩn FIPS 140, và module này có thể là một module mật mã khác với module thực hiện ML-DSA.Verify_internal.

Như đã lưu ý trong Mục 7.4, định danh liên kết với chữ ký nên chỉ ra liệu nên sử dụng ML-DSA hay phiên bản băm trước HashML-DSA cho quá trình xác minh chữ ký, cũng như hàm băm hoặc XOF sẽ được sử dụng để tính toán giá trị băm trước. Một chuỗi ngữ cảnh không rỗng nên được sử dụng trong quá trình xác minh nếu nó được chỉ định thông qua định danh của chữ ký hoặc bởi ứng dụng đang sử dụng chữ ký đó.

Thuật toán 5 HashML – DSA.Verify(pk, M, σ, ctx, PH) – *Xác minh một chữ ký HashML-DSA dạng băm trước*

Đầu vào: Khóa công khai $pk \in \mathbb{B}^{32+32k(\text{bitlen}(q-1)-d)}$, thông điệp $M \in \{0,1\}^*$, Chữ ký $\sigma \in \mathbb{B}^{\lambda/4+\ell \cdot 32 \cdot (1+\text{bitlen}(\gamma_1-1))+\omega+k}$, chuỗi ngữ cảnh ctx (một chuỗi byte có độ dài 255 byte hoặc ít hơn), hàm băm trước PH.

Đầu ra: Giá trị Boolean

```

1: if  $|ctx| > 255$  then
2:     return false                                ▷ trả về chỉ báo lỗi nếu chuỗi ngữ cảnh quá dài
3: end if
4:
5: switch PH do
6:     case SHA – 256:
7:         OID  $\leftarrow$  0x06, 0x09, 0x60, 0x86, 0x48, 0x01, 0x65, 0x03, 0x04, 0x02, 0x01
                                                    ▷ 2.16.840.1.101.3.4.2.1
8:          $\text{PH}_M \leftarrow \text{SHA256}(M)$ 
9:     case SHA – 512:
10:        OID  $\leftarrow$  0x06, 0x09, 0x60, 0x86, 0x48, 0x01, 0x65, 0x03, 0x04, 0x02, 0x03
                                                    ▷ 2.16.840.1.101.3.4.2.3
11:         $\text{PH}_M \leftarrow \text{SHA512}(M)$ 
12:     case SHAKE128:
13:        OID  $\leftarrow$  0x06, 0x09, 0x60, 0x86, 0x48, 0x01, 0x65, 0x03, 0x04, 0x02, 0x0B

```

```

14:       $PH_M \leftarrow \text{SHAKE128}(M, 256)$ 
15:      case ...
16:      ...
17: end switch
18:  $M' \leftarrow \text{BytesToBits}(\text{IntegerToBytes}(1,1) \parallel \text{IntegerToBytes}(|ctx|, 1) \parallel ctx \parallel \text{OID} \parallel PH_M)$ 
19: return ML-DSA.Verify_internal( $pk, M', \sigma$ )

```

8. Các hàm bên trong

Mục này mô tả các hàm cho việc tạo khóa, tạo chữ ký, và xác minh chữ ký ML-DSA. Ở những nơi yêu cầu tính ngẫu nhiên, các giá trị ngẫu nhiên được cung cấp làm đầu vào cho các hàm. Các giao diện được chỉ định trong mục này sẽ được sử dụng khi việc kiểm thử các bản cài đặt ML-DSA được thực hiện thông qua CAVP.

Ngoại trừ mục đích kiểm thử, các giao diện cho việc tạo khóa và tạo chữ ký được chỉ định trong mục này không nên được cung cấp cho các ứng dụng, vì bất kỳ giá trị ngẫu nhiên nào cần thiết cho việc tạo khóa và tạo chữ ký phải được tạo ra bởi module mật mã. Mục 7 cung cấp hướng dẫn về các giao diện sẽ được cung cấp cho các ứng dụng.

8.1. Tạo khóa ML-DSA (bên trong)

Thuật toán tạo khóa bên trong ML-DSA.KeyGen_internal nhận một mầm ngẫu nhiên 32-byte làm đầu vào và đưa ra một khóa công khai và một khóa bí mật, cả hai đều được mã hóa dưới dạng chuỗi byte.

Mầm ξ được mở rộng khi cần thiết bằng cách sử dụng một XOF (tức là, một biến thể byte của SHAKE256) được ký hiệu là H để sinh các giá trị ngẫu nhiên khác⁵. Cụ thể:

- Một mầm ngẫu nhiên công khai 32-byte ρ . Sử dụng mầm này, một ma trận đa thức $\mathbf{A} \in R_q^{k \times \ell}$ được lấy mẫu giả ngẫu nhiên⁶ từ $R_q^{k \times \ell}$.
- Một mầm ngẫu nhiên bí mật 64 -byte ρ' . Sử dụng mầm này, các vector đa thức $\mathbf{s}_1 \in R_q^\ell$ và $\mathbf{s}_2 \in R_q^k$ được lấy mẫu giả ngẫu nhiên từ tập con các vector đa thức mà các đa thức tọa độ của chúng có hệ số ngắn (tức là, nằm trong khoảng $[-\eta, \eta]$).
- Một mầm ngẫu nhiên bí mật 32 -byte K để sử dụng trong quá trình ký.

Phép toán mật mã cốt lõi tính toán giá trị công khai

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

⁵ Các mã hóa một byte của các tham số k và ℓ được đưa vào đầu vào XOF để phân tách miền. Đối với các bản triển khai sử dụng mầm thay cho khóa bí mật, điều này bảo đảm rằng việc mở rộng sẽ tạo ra một khóa không liên quan nếu mầm

⁶ Chính xác hơn, vì chỉ dạng NTT của $\mathbf{A}$, $\hat{\mathbf{A}} \in T_q^{k \times \ell} = \text{NTT}(\mathbf{A})$, là cần thiết trong các phép tính tiếp theo, nên mã thực tế tính $\hat{\mathbf{A}}$ như một mẫu giả ngẫu nhiên trên $T_q^{k \times \ell}$, còn việc lấy mẫu $\mathbf{A} = \text{NTT}^{-1}(\hat{\mathbf{A}})$ chỉ được thực hiện ngầm định (tức là có thể tính nhưng không được tính).

Vector $\mathbf{t}$ cùng với ma trận $\mathbf{A}$ có thể được coi là một dạng mở rộng của khóa công khai. Vector $\mathbf{t}$ được nén trong khóa công khai thực tế bằng cách loại bỏ d bit ít quan trọng nhất từ mỗi hệ số, từ đó tạo ra vector đa thức $\mathbf{t}_1$. Việc nén này là một sự tối ưu hóa về mặt hiệu suất, không phải về bảo mật. Các bit bậc thấp của $\mathbf{t}$ có thể được tái tạo lại từ một số lượng nhỏ các chữ ký và do đó không cần phải được coi là bí mật.

Khóa công khai ML-DSA pk là một mã hóa byte của mầm ngẫu nhiên công khai ρ và vector đa thức đã nén $\mathbf{t}_1$.

Khóa bí mật ML-DSA sk là một mã hóa byte của mầm ngẫu nhiên công khai ρ , một mầm ngẫu nhiên bí mật K để sử dụng trong quá trình ký, một giá trị băm 64-byte tr của khóa công khai để sử dụng trong quá trình ký, các vector đa thức bí mật $\mathbf{s}_1$ và $\mathbf{s}_2$, và một vector đa thức $\mathbf{t}_0$ mã hóa d bit có trọng số thấp nhất của mỗi hệ số thuộc đa thức khóa công khai chưa nén $\mathbf{t}$.

Thuật toán 6 ML-DSA.KeyGen_internal(ξ) – Tạo một cặp khóa công khai-khóa bí mật từ một mầm

Đầu vào: Mầm $\xi \in \mathbb{B}^{32}$.

Đầu ra: Khóa công khai $pk \in \mathbb{B}^{32+32k(\text{bitlen}(q-1)-d)}$ và khóa bí mật $sk \in \mathbb{B}^{32+32+64+32 \cdot ((\ell+k) \cdot \text{bitlen}(2\eta)+dk)}$

- 1: $(\rho, \rho', K) \in \mathbb{B}^{32} \times \mathbb{B}^{64} \times \mathbb{B}^{32} \leftarrow H(\xi \parallel \text{IntegerToBytes}(k, 1) \parallel \text{IntegerToBytes}(\ell, 1), 128)$
- 2: ▷ mở rộng mầm
- 3: $\hat{\mathbf{A}} \leftarrow \text{ExpandA}(\rho)$ ▷ $\hat{\mathbf{A}}$ được sinh và lưu trữ ở dạng biểu diễn NTT $\hat{\mathbf{A}}$
- 4: $(\mathbf{s}_1, \mathbf{s}_2) \leftarrow \text{ExpandS}(\rho')$
- 5: $\mathbf{t} \leftarrow \text{NTT}^{-1}(\hat{\mathbf{A}} \circ \text{NTT}(\mathbf{s}_1)) + \mathbf{s}_2$ ▷ Tính toán $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$
- 6: $(\mathbf{t}_1, \mathbf{t}_0) \leftarrow \text{Power2Round}(\mathbf{t})$ ▷ Nén $\mathbf{t}$
- 7: ▷ Power2Round được áp dụng theo từng thành phần (xem phần giải thích trong Mục 9.4)
- 8: $pk \leftarrow \text{pkEncode}(\rho, \mathbf{t}_1)$
- 9: $tr \leftarrow H(pk, 64)$
- 10: $sk \leftarrow \text{skEncode}(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$ ▷ K và tr được sử dụng khi sinh chữ ký
- 11: **return** (pk, sk)

8.2. Quá trình ký ML-DSA (bên trong)

ML-DSA.Sign_internal đưa ra một chữ ký được mã hóa dưới dạng chuỗi byte. Nó nhận đầu vào là một khóa bí mật sk được mã hóa dưới dạng chuỗi byte, một thông điệp đã được định dạng M' được mã hóa dưới dạng chuỗi bit, và một chuỗi 32 byte rnd . Có hai cách mà một thuật toán ký có thể sử dụng ML-DSA.Sign_internal: "có dự phòng" và "tắt định". Các biến thể "có dự phòng" mặc định của ML-DSA.Sign và HashML-DSA.Sign sử dụng một giá trị ngẫu nhiên mới cho rnd , trong khi các biến thể tắt định tùy chọn sử dụng chuỗi byte hằng số $\{0\}^{32}$ (xem Mục 5).

Trong cả hai biến thể, người ký trước tiên trích xuất các thành phần sau từ khóa bí mật: mầm ngẫu nhiên công khai ρ , mầm ngẫu nhiên bí mật 32 byte K , giá trị băm 64 byte của khóa công khai tr , các vectơ đa thức bí mật s_1 và s_2 , và vectơ đa thức t_0 mã hóa d bit có trọng số thấp nhất của mỗi hệ số thuộc đa thức khóa công khai chưa nén t . ρ sau đó được mở rộng thành cùng một ma trận A giống như trong quá trình tạo khóa.

Trước khi thông điệp M' được ký, nó được ghép nối với giá trị băm của khóa công khai tr và được băm xuống thành một đại diện thông điệp 64 byte μ bằng cách sử dụng H .

Người ký tạo ra một mầm 64 byte bổ sung ρ'' cho tính ngẫu nhiên bí mật trong mỗi thao tác ký. ρ'' được tính bằng $\rho'' \leftarrow H(K || rnd || \mu, 64)$. Trong biến thể có dự phòng (hedged) mặc định, rnd là đầu ra của một RBG, trong khi ở biến thể tắt định, rnd là một chuỗi 32 byte bao gồm toàn các số 0. Đây là điểm khác biệt duy nhất giữa biến thể tắt định và biến thể có dự phòng của ML-DSA.Sign.

Phần chính của thuật toán ký bao gồm một vòng lặp lấy mẫu loại bỏ trong đó mỗi lần lặp lại sẽ tạo ra một chữ ký hợp lệ hoặc một chữ ký không hợp lệ mà nếu tiết lộ nó sẽ làm rò rỉ thông tin về khóa bí mật. Vòng lặp được lặp lại cho đến khi tạo ra được một chữ ký hợp lệ, chữ ký này sau đó có thể được mã hóa dưới dạng một chuỗi byte và đưa ra. Vòng lặp lấy mẫu loại bỏ tuân theo khung kỹ thuật Fiat-Shamir With Aborts [10] và (ngoại trừ bước loại bỏ) có cấu trúc tương tự như chữ ký Schnorr [30] (ví dụ: EdDSA [31]). Người ký trước tiên tạo ra một “cam kết” w_1 và sau đó dẫn xuất giả ngẫu nhiên một “thử thách” c từ w_1 và đại diện thông điệp μ . Cuối cùng, người ký tính toán một phản hồi z .

Chi tiết hơn, các tính toán chính liên quan trong vòng lặp lấy mẫu loại bỏ diễn ra như sau:

- Sử dụng hàm ExpandMask (**Thuật toán 34**), mầm ρ'' , và một bộ đếm κ , một vectơ đa thức $y \in R_q^\ell$ được lấy mẫu giả ngẫu nhiên từ tập con các vectơ đa thức có các hệ số tương đối ngắn (tức là, trong khoảng $[-\gamma_1 + 1, \gamma_1]$).
- Từ y , người ký tính toán cam kết w_1 bằng cách tính $w = Ay$ và sau đó làm tròn đến một bội số gần nhất của $2\gamma_2$ bằng cách sử dụng HighBits (**Thuật toán 37**).
- w_1 và μ được ghép nối và băm để tạo ra giá trị băm cam kết $\tilde{c}$. Quá trình này sử dụng hàm w1Encode (**Thuật toán 28**). Chuỗi byte $\tilde{c}$ được sử dụng để lấy mẫu giả ngẫu nhiên một đa thức $c \in R_q$ có các hệ số thuộc $\{-1, 0, 1\}$ và trọng số Hamming là τ . Việc lấy mẫu được thực hiện bằng hàm SampleInBall (**Thuật toán 29**).
- Người ký tính toán phản hồi $z = y + cs_1$ và thực hiện các kiểm tra tính hợp lệ khác nhau. Nếu bất kỳ kiểm tra nào thất bại, người ký sẽ tiếp tục vòng lặp lấy mẫu loại bỏ.
- Nếu các kiểm tra thành công, người ký có thể tính toán một đa thức gợi ý h , điều này sẽ cho phép bên xác minh khôi phục lại w_1 bằng cách sử dụng khóa công khai đã nén cùng với các thành phần khác của chữ ký. Quá trình này sử dụng hàm MakeHint (**Thuật toán 39**). Người ký sau đó sẽ đưa ra chữ ký cuối cùng, là một mã hóa dạng byte của giá trị băm cam kết $\tilde{c}$, phản hồi z , và đa thức gợi ý h .

Ngoài ra, có một cách khác để cài đặt các kiểm tra tính hợp lệ trên z và tính toán h , được mô tả trong Mục

5.1 của [6]. Phương pháp này cũng có thể được sử dụng trong các bản cài đặt của ML-DSA.

Trong **Thuật toán 7**, các biến đôi khi được sử dụng để lưu trữ các tích nhằm tránh việc phải tính toán lại chúng sau này trong thuật toán ký. Các tích được tính toán trước này được biểu thị trong giả mã bằng một cặp dấu ngoặc góc kép bao quanh các biến được nhân (ví dụ: $\langle\langle cs_1 \rangle\rangle$).

Thuật toán 7 ML-DSA.Sign_internal(sk, M', rnd) – Thuật toán tắt định để tạo chữ ký cho một thông điệp đã được định dạng M'

Đầu vào: Khóa bí mật $sk \in \mathbb{B}^{32+32+64+32 \cdot ((\ell+k) \cdot \text{bitlen}(2\eta)+dk)}$, thông điệp đã định dạng $M' \in \{0,1\}^*$, và giá trị ngẫu nhiên cho mỗi thông điệp hoặc biến giả $rnd \in \mathbb{B}^{32}$.

Đầu ra: Chữ ký $\sigma \in \mathbb{B}^{\lambda/4+\ell \cdot 32 \cdot (1+\text{bitlen}(\gamma_1-1))+\omega+k}$.

- 1: $(\rho, K, tr, s_1, s_2, t_0) \leftarrow \text{skDecode}(sk)$
- 2: $\hat{s}_1 \leftarrow \text{NTT}(s_1)$
- 3: $\hat{s}_2 \leftarrow \text{NTT}(s_2)$
- 4: $\hat{t}_0 \leftarrow \text{NTT}(t_0)$
- 5: $\hat{A} \leftarrow \text{ExpandA}(\rho)$ ▷ A được tạo và lưu trữ dưới dạng biểu diễn NTT là $\hat{A}$
- 6: $\mu \leftarrow H(\text{BytesToBits}(tr) \| M', 64)$ ▷ đại diện thông điệp, có thể tùy chọn
được tính trong một mô-đun mật mã khác
- 7: $\rho'' \leftarrow H(K \| rnd \| \mu, 64)$ ▷ tính mầm ngẫu nhiên bí mật
- 8: $\kappa \leftarrow 0$ ▷ khởi tạo bộ đếm κ
- 9: $(z, h) \leftarrow \perp$
- 10: **while** $(z, h) = \perp$ **do** ▷ vòng lặp lấy mẫu loại bỏ
- 11: $y \in R_q^\ell \leftarrow \text{ExpandMask}(\rho'', \kappa)$
- 12: $w \leftarrow \text{NTT}^{-1}(\hat{A} \circ \text{NTT}(y))$
- 13: $w_1 \leftarrow \text{HighBits}(w)$ ▷ cam kết của người ký
- 14: ▷ HighBits được áp dụng cho từng thành phần (xem giải thích ở Mục 9.4)
- 15: $\tilde{c} \leftarrow H(\mu \| w_1 \text{Encode}(w_1), \lambda/4)$ ▷ giá trị băm cam kết
- 16: $c \in R_q \leftarrow \text{SampleInBall}(\tilde{c})$ ▷ thử thách của bên xác minh
- 17: $\hat{c} \leftarrow \text{NTT}(c)$
- 18: $\langle\langle cs_1 \rangle\rangle \leftarrow \text{NTT}^{-1}(\hat{c} \circ \hat{s}_1)$
- 19: $\langle\langle cs_2 \rangle\rangle \leftarrow \text{NTT}^{-1}(\hat{c} \circ \hat{s}_2)$
- 20: $z \leftarrow y + \langle\langle cs_1 \rangle\rangle$ ▷ phản hồi của người ký
- 21: $r_0 \leftarrow \text{LowBits}(w - \langle\langle cs_2 \rangle\rangle)$
- 22: ▷ LowBits được áp dụng cho từng thành phần (xem giải thích ở Mục 9.4)

```

23:   if  $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$  or  $\|\mathbf{r}_0\|_\infty \geq \gamma_2 - \beta$  then  $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$  ▷ kiểm tra tính hợp lệ
24:   else
25:      $\langle \langle \mathbf{ct}_0 \rangle \rangle \leftarrow \text{NTT}^{-1}(\hat{c} \circ \hat{\mathbf{t}}_0)$ 
26:      $\mathbf{h} \leftarrow \text{MakeHint}(-\langle \langle \mathbf{ct}_0 \rangle \rangle, \mathbf{w} - \langle \langle \mathbf{cs}_2 \rangle \rangle + \langle \langle \mathbf{ct}_0 \rangle \rangle)$  ▷ gợi ý của người ký
27:     ▷ MakeHint được áp dụng cho từng thành phần (xem giải thích ở Mục 9.4)
28:     if  $\|\langle \langle \mathbf{ct}_0 \rangle \rangle\|_\infty \geq \gamma_2$  or số lượng giá trị 1 trong  $\mathbf{h}$  lớn hơn  $\omega$ , then  $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$ 
29:     end if
30:   end if
31:    $\kappa \leftarrow \kappa + \ell$  ▷ tăng bộ đếm
32: end while
33:  $\sigma \leftarrow \text{sigEncode}(\tilde{c}, \mathbf{z} \bmod^\pm q, \mathbf{h})$ 
34: return  $\sigma$ 

```

8.3. Xác minh ML-DSA (Bên trong)

Thuật toán ML-DSA.Verify_internal (**Thuật toán 8**) nhận đầu vào là một khóa công khai pk được mã hóa dưới dạng chuỗi byte, một thông điệp M' được mã hóa dưới dạng chuỗi bit, và một chữ ký σ được mã hóa dưới dạng chuỗi byte. Không có yếu tố ngẫu nhiên nào được yêu cầu đối với ML-DSA.Verify_internal. Nó tạo ra một giá trị Boolean (tức là, một giá trị là true nếu chữ ký là hợp lệ đối với thông điệp và khóa công khai, và false nếu chữ ký không hợp lệ) làm đầu ra. **Thuật toán 8** chỉ định độ dài của chữ ký σ và khóa công khai pk theo các tham số được mô tả trong Bảng 1. Nếu một bản cài đặt của ML-DSA.Verify_internal có thể chấp nhận đầu vào cho σ hoặc pk với bất kỳ độ dài nào khác, nó phải trả về false bất cứ khi nào độ dài của một trong hai đầu vào này khác với độ dài được chỉ định của nó.

Bên xác minh trước tiên trích xuất mầm ngẫu nhiên công khai ρ và vectơ đa thức đã nén $\mathbf{t}_1$ từ khóa công khai pk , sau đó trích xuất giá trị băm cam kết của người ký $\tilde{c}$, phản hồi $\mathbf{z}$, và gợi ý $\mathbf{h}$ từ chữ ký σ . Bên xác minh có thể nhận thấy rằng gợi ý không được mã hóa byte đúng cách, được biểu thị bằng ký hiệu " $\perp$ ", trong trường hợp đó, thuật toán xác minh sẽ ngay lập tức trả về false để chỉ ra rằng chữ ký không hợp lệ.

Giả sử rằng chữ ký được trích xuất thành công từ mã hóa byte của nó, bên xác minh dẫn xuất giả ngẫu nhiên $\mathbf{A}$ từ ρ , giống như những gì được thực hiện trong quá trình tạo khóa và ký, và tạo ra một đại diện thông điệp μ bằng cách băm chuỗi ghép nối của tr (tức là, giá trị băm của khóa công khai pk) và thông điệp M . Bên xác minh sau đó cố gắng khôi phục lại cam kết của người ký (tức là, vectơ đa thức $\mathbf{w}_1$) từ khóa công khai pk và chữ ký σ . Trong ML-DSA.Sign_internal, $\mathbf{w}_1$ được tính bằng cách làm tròn $\mathbf{w} = \mathbf{A}\mathbf{y}$. Trong ML-DSA.Verify_internal, giá trị được khôi phục của $\mathbf{w}_1$ được gọi là $\mathbf{w}'_1$ vì nó có thể đã được tính toán theo một cách khác nếu chữ ký không hợp lệ. Giá trị $\mathbf{w}'_1$ này được tính toán thông qua quy trình sau:

- Dẫn xuất đa thức thử thách c từ giá trị băm cam kết của người ký $\tilde{c}$, hoàn toàn tương tự như cách được thực hiện trong ML-DSA.Sign_internal.
- Sử dụng phản hồi của người ký z để tính toán

$$\mathbf{w}'_{\text{Approx}} = \mathbf{Az} - c\mathbf{t}_1 \cdot 2^d.$$

Giả sử chữ ký được tính toán chính xác, như trong ML-DSA.Sign_internal, ta có

$$\mathbf{w} = \mathbf{Ay} = \mathbf{Az} - c\mathbf{t} + c\mathbf{s}_2 \approx \mathbf{w}'_{\text{Approx}} = \mathbf{Az} - c\mathbf{t}_1 \cdot 2^d$$

bởi vì c và $\mathbf{s}_2$ có các hệ số nhỏ, và $\mathbf{t}_1 \cdot 2^d \approx \mathbf{t}$.

- Sử dụng gợi ý của người ký $\mathbf{h}$ để thu được $\mathbf{w}'_1$ từ $\mathbf{w}'_{\text{Approx}}$.

Cuối cùng, bên xác minh kiểm tra xem phản hồi của người ký z và gợi ý của người ký $\mathbf{h}$ có hợp lệ hay không và $\mathbf{w}'_1$ được khôi phục có nhất quán với giá trị băm cam kết của người ký $\tilde{c}$ hay không. Chính xác hơn, bên xác minh kiểm tra xem tất cả các hệ số của z có đủ nhỏ hay không (tức là, trong khoảng $(-(\gamma_1 - \beta), \gamma_1 - \beta)$), $\mathbf{h}$ chứa không quá ω hệ số khác 0, và $\tilde{c}$ khớp với giá trị băm $\tilde{c}'$ của đại diện thông điệp μ ghép nối với $\mathbf{w}'_1$ (được biểu diễn dưới dạng chuỗi byte). Nếu tất cả các kiểm tra này thành công, thì ML-DSA.Verify_internal trả về true. Ngược lại, nó trả về false.

Thuật toán 8 ML – DSA.Verify_internal(pk, M', σ) – Hàm nội bộ để xác minh chữ ký σ cho một thông điệp đã được định dạng M'

Đầu vào: Khóa công khai $pk \in \mathbb{B}^{32+32k(\text{bitlen}(q-1)-d)}$, và thông điệp $M' \in \{0,1\}^*$.

Đầu vào: Chữ ký $\sigma \in \mathbb{B}^{\lambda/4+\ell \cdot 32 \cdot (1+\text{bitlen}(\gamma_1-1))+\omega+k}$

Đầu ra: Giá trị Boolean

- 1: $(\rho, \mathbf{t}_1) \leftarrow \text{pkDecode}(pk)$
- 2: $(\tilde{c}, \mathbf{z}, \mathbf{h}) \leftarrow \text{sigDecode}(\sigma)$ ▷ giá trị băm cam kết $\tilde{c}$, phản hồi $\mathbf{z}$ và gợi ý $\mathbf{h}$ của người ký
- 3: **if** $\mathbf{h} = \perp$ **then return** false ▷ gợi ý không được mã hóa đúng
- 4: **end if**
- 5: $\hat{\mathbf{A}} \leftarrow \text{ExpandA}(\rho)$ ▷ $\mathbf{A}$ được sinh và lưu ở dạng biểu diễn NTT $\hat{\mathbf{A}}$
- 6: $tr \leftarrow H(pk, 64)$
- 7: $\mu \leftarrow H(\text{BytesToBits}(tr) \| M', 64)$ ▷ đại diện của thông điệp, có thể tùy chọn tính trong một mô-đun mật mã khác
- 8: $c \in R_q \leftarrow \text{SampleInBall}(\tilde{c})$ ▷ Tính thử thách của bên xác minh từ $\tilde{c}$
- 9: $\mathbf{w}'_{\text{Approx}} \leftarrow \text{NTT}^{-1}(\hat{\mathbf{A}} \circ \text{NTT}(\mathbf{z}) - \text{NTT}(c) \circ \text{NTT}(\mathbf{t}_1 \cdot 2^d))$ ▷ $\mathbf{w}'_{\text{Approx}} = \mathbf{Az} - c\mathbf{t}_1 \cdot 2^d$
- 10: $\mathbf{w}'_1 \leftarrow \text{UseHint}(\mathbf{h}, \mathbf{w}'_{\text{Approx}})$ ▷ Khôi phục cam kết của bên ký
- 11: ▷ UseHint được áp dụng theo từng thành phần (xem giải thích ở Mục 9.4)
- 12: $\tilde{c}' \leftarrow H(\mu \| \text{w1Encode}(\mathbf{w}'_1), \lambda/4)$ ▷ băm giá trị này; kết quả phải khớp với $\tilde{c}$

13: **return** $\llbracket \lVert \mathbf{z} \rVert_\infty < \gamma_1 - \beta \rrbracket$ **and** $\llbracket \tilde{c} = \tilde{c}' \rrbracket$

9. Các hàm hỗ trợ

Phần này cung cấp giả mã cho các chương trình con được sử dụng bởi ML-DSA, bao gồm các hàm chuyển đổi kiểu dữ liệu, số học và lấy mẫu.

9.1. Chuyển đổi giữa các kiểu dữ liệu

Mặc dù kiểu dữ liệu chính trong ML-DSA là chuỗi byte, các kiểu dữ liệu khác cũng được sử dụng. Mục tiêu trong phần này là xây dựng các thủ tục để chuyển đổi giữa các đối tượng đại số khác nhau được định nghĩa trong Mục 4. Các **Thuật toán 9-13** là các thủ tục trung gian để chuyển đổi giữa chuỗi bit, chuỗi byte và số nguyên.

Thuật toán 9 IntegerToBits(x, α) - Tính biểu diễn cơ số 2 của $x \bmod 2^\alpha$ theo thứ tự little-endian

Đầu vào: Một số nguyên không âm x và một số nguyên dương α

Đầu ra: Một chuỗi bit y có độ dài α

1: $x' \leftarrow x$

2: **for** i **from** 0 **to** $\alpha - 1$ **do**

3: $y[i] \leftarrow x' \bmod 2$

4: $x' \leftarrow \lfloor x'/2 \rfloor$

5: **end for**

6: **return** y

Thuật toán 10 BitsToInteger(y, α) – Tính giá trị số nguyên được biểu diễn bởi một chuỗi bit theo thứ tự little-endian

Đầu vào: Một số nguyên dương α và một chuỗi bit y có độ dài α

Đầu ra: Một số nguyên không âm x .

1: $x \leftarrow 0$

2: **for** i **from** 1 **to** α **do**

3: $x \leftarrow 2x + y[\alpha - i]$

4: **end for**

5: **return** x

Thuật toán 11 IntegerToBytes(x, α) – Tính biểu diễn cơ số 256 của $x \bmod 256^\alpha$ theo thứ tự little-endian

Đầu vào: Một số nguyên không âm x và một số nguyên dương α

Đầu ra: Một chuỗi byte y có độ dài α

```

1:  $x' \leftarrow x$ 
2: for  $i$  from 0 to  $\alpha - 1$  do
3:    $y[i] \leftarrow x' \bmod 256$ 
4:    $x' \leftarrow \lfloor x' / 256 \rfloor$ 
5: end for
6: return  $y$ 

```

Thuật toán 12 BitsToBytes(y) – Chuyển đổi một chuỗi bit thành một chuỗi byte theo thứ tự little-endian

Đầu vào: Một chuỗi bit y có độ dài α

Đầu ra: Một chuỗi byte z có độ dài $\lceil \alpha / 8 \rceil$

```

1:  $z \in \mathbb{B}^{\lceil \alpha / 8 \rceil} \leftarrow 0^{\lceil \alpha / 8 \rceil}$ 
2: for  $i$  from 0 to  $\alpha - 1$  do
3:    $z[\lfloor i / 8 \rfloor] \leftarrow z[\lfloor i / 8 \rfloor] + y[i] \cdot 2^{i \bmod 8}$ 
4: end for
5: return  $z$ 

```

Thuật toán 13 BytesToBits(z) – Chuyển đổi một chuỗi byte thành một chuỗi bit theo thứ tự little-endian

Đầu vào: Một chuỗi byte z có độ dài α .

Đầu ra: Một chuỗi bit y có độ dài 8α .

```

1:  $z' \leftarrow z$ 
2: for  $i$  from 0 to  $\alpha - 1$  do
3:   for  $j$  from 0 to 7 do                                     ▷ chuyển đổi byte  $z[i]$  thành 8 bit
4:      $y[8i + j] \leftarrow z'[i] \bmod 2$ 
5:      $z'[i] \leftarrow \lfloor z'[i] / 2 \rfloor$ 
6:   end for
7: end for
8: return  $y$ 

```

Các **thuật toán 14-15** chuyển đổi các chuỗi byte thành các hệ số của đa thức trong R . CoeffFromThreeBytes

sử dụng một chuỗi 3 byte để sinh ra một phần tử thuộc $\{0,1,\dots,q-1\}$ hoặc trả về ký hiệu rỗng $\perp$. CoeffFromHalfByte sử dụng một phần tử thuộc $\{0,1,\dots,15\}$ để sinh ra một phần tử thuộc $\{-\eta,-\eta+1,\dots,\eta\}$ hoặc trả về $\perp$. Hai thủ tục này sẽ được sử dụng trong các thuật toán lấy mẫu đều RejNTTPoly và RejBoundedPoly, được thảo luận trong Mục 9.3.

Thuật toán 14 CoeffFromThreeBytes(b_0, b_1, b_2) – Sinh ra một phần tử thuộc $\{0,1,2,\dots,q-1\} \cup \{\perp\}$	
Đầu vào: Các byte b_0, b_1, b_2	
Đầu ra: Một số nguyên theo modulo q hoặc $\perp$.	
1: $b'_2 \leftarrow b_2$	
2: if $b'_2 > 127$ then	
3: $b'_2 \leftarrow b'_2 - 128$	▷ thiết lập bit cao nhất của b'_2 bằng không
4: end if	
5: $z \leftarrow 2^{16} \cdot b'_2 + 2^8 \cdot b_1 + b_0$	▷ $0 \leq z \leq 2^{23} - 1$
6: if $z < q$ then return z	▷ lấy mẫu loại bỏ
7: else return $\perp$	
8: end if	

Thuật toán 15 CoeffFromHalfByte(b) – Gọi $\eta \in \{2,4\}$. Sinh ra một phần tử thuộc $\{-\eta,-\eta+1,\dots,\eta\} \cup \{\perp\}$.	
Đầu vào: Số nguyên $b \in \{0,1,\dots,15\}$	
Đầu ra: Một số nguyên thuộc $[-\eta,\eta]$, hoặc $\perp$	
1: if $\eta = 2$ and $b < 15$ then return $2 - (b \bmod 5)$	▷ lấy mẫu loại bỏ từ $\{-2,\dots,2\}$
2: else	
3: if $\eta = 4$ and $b < 9$ then return $4 - b$	▷ lấy mẫu loại bỏ từ $\{-4,\dots,4\}$
4: else return $\perp$	
5: end if	
6: end if	

Các **Thuật toán 16-19** chuyển đổi một cách hiệu quả một phần tử $w \in R$ thành một chuỗi byte và ngược lại với giả thiết rằng các hệ số của w nằm trong một khoảng bị giới hạn. SimpleBitPack giả định rằng $w_i \in [0, b]$ đối với một số nguyên dương b nào đó và đóng gói w thành một chuỗi byte có độ dài $32 \cdot \text{bitlen } b$. BitPack cho phép điều kiện giới hạn tổng quát hơn là $w_i \in [-a, b]$. Thuật toán BitPack hoạt động đơn giản bằng cách lấy đa thức $\sum_{i=0}^{255} bX^i$ trừ đi w .

Thuật toán 16 SimpleBitPack(w, b) – Mã hóa một đa thức w thành một chuỗi byte
--

Đầu vào: $b \in \mathbb{N}$ và $w \in R$ sao cho tất cả các hệ số của w đều nằm trong đoạn $[0, b]$.

Đầu ra: Một chuỗi byte có độ dài $32 \cdot \text{bitlen } b$.

```

1:  $z \leftarrow ()$  ▷ thiết lập  $z$  là chuỗi bit rỗng
2: for  $i$  from 0 to 255 do
3:    $z \leftarrow z \parallel \text{IntegerToBits}(w_i, \text{bitlen } b)$ 
4: end for
5: return  $\text{BitsToBytes}(z)$ 

```

Thuật toán 17 $\text{BitPack}(w, a, b)$ – Mã hóa một đa thức w thành một chuỗi byte

Đầu vào: $a, b \in \mathbb{N}$ và $w \in R$ sao cho tất cả các hệ số của w đều nằm trong đoạn $[-a, b]$.

Đầu ra: Một chuỗi byte có độ dài $32 \cdot \text{bitlen } (a + b)$

```

1:  $z \leftarrow ()$  ▷ thiết lập  $z$  là chuỗi bit rỗng
2: for  $i$  from 0 to 255 do
3:    $z \leftarrow z \parallel \text{IntegerToBits}(b - w_i, \text{bitlen } (a + b))$ 
4: end for
5: return  $\text{BitsToBytes}(z)$ 

```

Các hàm SimpleBitUnpack và BitUnpack được sử dụng để giải mã các chuỗi byte được sinh ra bởi các hàm ở trên. Đối với một số lựa chọn của a và b , tồn tại các chuỗi byte sai định dạng khiến cho SimpleBitUnpack và BitUnpack đưa ra các đa thức có hệ số không nằm trong các khoảng tương ứng là $[0, b]$ và $[-a, b]$. Điều này có thể là một mối lo ngại khi chạy SimpleBitUnpack và BitUnpack trên các đầu vào có thể xuất phát từ một nguồn không đáng tin cậy.

Thuật toán 18 $\text{SimpleBitUnpack}(v, b)$ – Thực hiện ngược thủ tục SimpleBitPack

Đầu vào: $b \in \mathbb{N}$ và một chuỗi byte v có độ dài $32 \cdot \text{bitlen } b$.

Đầu ra: Một đa thức $w \in R$ với các hệ số nằm trong $[0, 2^c - 1]$, trong đó $c = \text{bitlen } b$. Khi $b + 1$ là lũy thừa của 2, các hệ số nằm trong $[0, b]$.

```

1:  $c \leftarrow \text{bitlen } b$ 
2:  $z \leftarrow \text{BytesToBits}(v)$ 
3: for  $i$  from 0 to 255 do
4:    $w_i \leftarrow \text{BitsToInteger}((z[ic], z[ic + 1], \dots, z[ic + c - 1]), c)$ 
5: end for

```

6: **return** w

Thuật toán 19 BitUnpack(v, a, b) – Thực hiện ngược thủ tục BitPack

Đầu vào: $a, b \in \mathbb{N}$ và một chuỗi byte v có độ dài $32 \cdot \text{bitlen}(a + b)$

Đầu ra: Một đa thức $w \in R$ với các hệ số nằm trong $[b - 2^c + 1, b]$, trong đó $c = \text{bitlen}(a + b)$

Khi $a + b + 1$ là một lũy thừa của 2, các hệ số nằm trong $[-a, b]$.

1: $c \leftarrow \text{bitlen}(a + b)$

2: $z \leftarrow \text{BytesToBits}(v)$

3: **for** i **from** 0 **to** 255 **do**

4: $w_i \leftarrow b - \text{BitsToInteger}((z[i \cdot c], z[i \cdot c + 1], \dots, z[i \cdot c + c - 1]), c)$

5: **end for**

6: **return** w

Các **Thuật toán 20** và **Thuật toán 21** thực hiện các phép chuyển đổi từ chuỗi byte sang đa thức đối với các đa thức có hệ số nhị phân thưa. Cụ thể, các thuật toán ký và xác minh (Các mục 8.2 và 8.3) sử dụng một "gọi ý", là một vector các đa thức $\mathbf{h} \in R_2^k$ sao cho tổng số các hệ số trong $\mathbf{h}[0], \mathbf{h}[1], \dots, \mathbf{h}[k - 1]$ có giá trị bằng 1 không vượt quá ω . Ràng buộc này cho phép các thủ tục mã hóa và giải mã hiệu quả hơn (mặc dù phức tạp hơn) so với BitPack và BitUnpack.

HintBitPack($\mathbf{h}$) đưa ra một chuỗi byte y có độ dài $\omega + k \cdot k$ byte cuối của y chứa thông tin về số lượng các hệ số khác không hiện diện trong mỗi đa thức $\mathbf{h}[0], \mathbf{h}[1], \dots, \mathbf{h}[k - 1]$, và ω byte đầu tiên của y chứa thông tin về vị trí chính xác mà các số hạng khác không đó xuất hiện. HintBitUnpack đảo ngược thủ tục được thực hiện bởi HintBitPack và khôi phục lại vector $\mathbf{h}$.

Thuật toán 20 HintBitPack($\mathbf{h}$) – Mã hóa một vector đa thức $\mathbf{h}$ có các hệ số nhị phân thành một chuỗi byte

Đầu vào: Một vector đa thức $\mathbf{h} \in R_2^k$ sao cho các đa thức $\mathbf{h}[0], \mathbf{h}[1], \dots, \mathbf{h}[k - 1]$ có tổng cộng nhiều nhất ω hệ số khác không.

Đầu ra: Một chuỗi byte y có độ dài $\omega + k$, mã hóa $\mathbf{h}$ như mô tả ở trên.

1: $y \in \mathbb{B}^{\omega+k} \leftarrow 0^{\omega+k}$

2: Index $\leftarrow 0$

▷ chỉ số để ghi ω byte đầu tiên của y

3: **for** i **from** 0 **to** $k - 1$ **do**

▷ xét $\mathbf{h}[i]$

4: **for** j **from** 0 **to** 255 **do**

5: **if** $\mathbf{h}[i]_j \neq 0$ **then**

6: $y[\text{Index}] \leftarrow j$

▷ lưu vị trí của các hệ số khác không trong $\mathbf{h}[i]$

7: Index $\leftarrow \text{Index} + 1$

```

8:     end if
9: end for
10:   $y[\omega + i] \leftarrow \text{Index}$                                 ▷ sau khi xử lý  $\mathbf{h}[i]$ , lưu giá trị của Index
11: end for
12: return  $y$ 

```

Thuật toán 21 HintBitUnpack(y) – Thực hiện ngược thủ tục HintBitPack

Đầu vào: Một chuỗi byte y có độ dài $\omega + k$ mã hóa $\mathbf{h}$ như mô tả ở trên.

Đầu ra: Một vector đa thức $\mathbf{h} \in R_2^k$ hoặc $\perp$.

```

1:  $\mathbf{h} \in R_2^k \leftarrow 0^k$ 
2: Index  $\leftarrow 0$                                 ▷ chỉ số để đọc  $\omega$  byte đầu tiên của  $y$ 
3: for  $i$  from 0 to  $k - 1$  do                            ▷ khôi phục  $\mathbf{h}[i]$ 
4:     if  $y[\omega + i] < \text{Index}$  or  $y[\omega + i] > \omega$  then return  $\perp$     ▷ đầu vào sai định dạng
5: end if
6: First  $\leftarrow \text{Index}$ 
7: while Index  $< y[\omega + i]$  do                            ▷  $y[\omega + i]$  cho biết có thể tăng Index đến đâu
8:     if Index  $> \text{First}$  then
9:         if  $y[\text{Index} - 1] \geq y[\text{Index}]$  then return  $\perp$                 ▷ đầu vào sai định dạng
10:        end if
11:    end if
12:     $\mathbf{h}[i]_{y[\text{Index}]} \leftarrow 1$                                 ▷  $y[\text{Index}]$  cho biết hệ số nào trong  $\mathbf{h}[i]$  nên bằng 1
13:    Index  $\leftarrow \text{Index} + 1$ 
14: end while
15: end for
16: for  $i$  from Index to  $\omega - 1$  do                            ▷ đọc các byte còn lại trong  $\omega$  byte đầu tiên của  $y$ 
17:     if  $y[i] \neq 0$  then return  $\perp$                                 ▷ đầu vào sai định dạng
18: end if
19: end for
20: return  $\mathbf{h}$ 

```

9.2. Mã hóa khóa và chữ ký ML-DSA

Các Thuật toán 22-27 chuyển đổi các khóa và chữ ký cho ML-DSA thành các chuỗi byte. Các thủ tục này nhận vào các dãy đối tượng đại số nhất định, mã hóa chúng liên tiếp thành các chuỗi byte, và thực hiện các thủ tục giải mã tương ứng.

Đầu tiên, pkEncode và pkDecode chuyển đổi các khóa công khai ML-DSA thành các chuỗi byte và ngược lại. Khi xác minh một chữ ký, pkDecode có thể được chạy trên một đầu vào xuất phát từ một nguồn không đáng tin cậy. Do đó, cần cẩn trọng khi sử dụng SimpleBitUnpack. Như được sử dụng ở đây, SimpleBitUnpack luôn trả về các giá trị nằm trong khoảng chính xác.

Thuật toán 22 $\text{pkEncode}(\rho, \mathbf{t}_1)$ – Mã hóa một khóa công khai cho ML-DSA thành một chuỗi byte
Đầu vào: $\rho \in \mathbb{B}^{32}, \mathbf{t}_1 \in R^k$ với các hệ số nằm trong $[0, 2^{\text{bitlen}(q-1)-d} - 1]$ Đầu ra: Khóa công khai $pk \in \mathbb{B}^{32+32k(\text{bitlen}(q-1)-d)}$ 1: $pk \leftarrow \rho$ 2: for i from 0 to $k - 1$ do 3: $pk \leftarrow pk \parallel \text{SimpleBitPack}(\mathbf{t}_1[i], 2^{\text{bitlen}(q-1)-d} - 1)$ 4: end for 5: return pk

Thuật toán 23 $\text{pkDecode}(pk)$ – Thực hiện ngược thủ tục pkEncode
Đầu vào: Khóa công khai $pk \in \mathbb{B}^{32+32k(\text{bitlen}(q-1)-d)}$. Đầu ra: $\rho \in \mathbb{B}^{32}, \mathbf{t}_1 \in R^k$ với các hệ số nằm trong $[0, 2^{\text{bitlen}(q-1)-d} - 1]$. 1: $(\rho, z_0, \dots, z_{k-1}) \in \mathbb{B}^{32} \times (\mathbb{B}^{32(\text{bitlen}(q-1)-d)})^k \leftarrow pk$ 2: for i from 0 to $k - 1$ do 3: $\mathbf{t}_1[i] \leftarrow \text{SimpleBitUnpack}(z_i, 2^{\text{bitlen}(q-1)-d} - 1)$ ▷ kết quả này luôn nằm trong khoảng đúng 4: end for 5: return $(\rho, \mathbf{t}_1)$

Tiếp theo, skEncode và skDecode chuyển đổi các khóa bí mật ML-DSA thành các chuỗi byte và ngược lại. Lưu ý rằng tồn tại các đầu vào sai định dạng có thể khiến skDecode trả về các giá trị không nằm trong khoảng đúng. Do đó, chỉ nên chạy skDecode trên các đầu vào đến từ các nguồn đáng tin cậy.

Thuật toán 24 $\text{skEncode}(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$ – Mã hóa một khóa bí mật cho ML-DSA thành một chuỗi byte
Đầu vào: $\rho \in \mathbb{B}^{32}, K \in \mathbb{B}^{32}, tr \in \mathbb{B}^{64}, \mathbf{s}_1 \in R^\ell$ với các hệ số nằm trong $[-\eta, \eta], \mathbf{s}_2 \in R^k$ với các hệ số nằm trong $[-\eta, \eta], \mathbf{t}_0 \in R^k$ với các hệ số nằm trong $[-2^{d-1} + 1, 2^{d-1}]$ Đầu ra: Khóa bí mật $sk \in \mathbb{B}^{32+32+64+32 \cdot ((k+\ell) \cdot \text{bitlen}(2\eta) + dk)}$ 1: $sk \leftarrow \rho \parallel K \parallel tr$ 2: for i from 0 to $\ell - 1$ do

```

3:    $sk \leftarrow sk \parallel \text{BitPack}(s_1[i], \eta, \eta)$ 
4: end for
5: for  $i$  from 0 to  $k-1$  do
6:    $sk \leftarrow sk \parallel \text{BitPack}(s_2[i], \eta, \eta)$ 
7: end for
8: for  $i$  from 0 to  $k-1$  do
9:    $sk \leftarrow sk \parallel \text{BitPack}(t_0[i], 2^{d-1} - 1, 2^{d-1})$ 
10: end for
11: return  $sk$ 

```

Thuật toán 25 $\text{skDecode}(sk)$ – Thực hiện ngược thủ tục skEncode

Đầu vào: Khóa bí mật $sk \in \mathbb{B}^{32+32+64+32 \cdot ((k+\ell) \cdot \text{bitlen}(2\eta) + dk)}$.

Đầu ra: $\rho \in \mathbb{B}^{32}, K \in \mathbb{B}^{32}, tr \in \mathbb{B}^{64}, s_1 \in R^\ell, s_2 \in R^k, t_0 \in R^k$ với các hệ số nằm trong $[-2^{d-1} + 1, 2^{d-1}]$

1: $(\rho, K, tr, y_0, \dots, y_{\ell-1}, z_0, \dots, z_{k-1}, w_0, \dots, w_{k-1}) \in \mathbb{B}^{32} \times \mathbb{B}^{32} \times \mathbb{B}^{64} \times (\mathbb{B}^{32 \cdot \text{bitlen}(2\eta)})^\ell \times (\mathbb{B}^{32 \cdot \text{bitlen}(2\eta)})^k \times (\mathbb{B}^{32d})^k \leftarrow sk$

2: **for** i **from** 0 **to** $\ell - 1$ **do**

3: $s_1[i] \leftarrow \text{BitUnpack}(y_i, \eta, \eta)$ ▷ kết quả này có thể nằm ngoài $[-\eta, \eta]$ nếu đầu vào sai định dạng

4: **end for**

5: **for** i **from** 0 **to** $k - 1$ **do**

6: $s_2[i] \leftarrow \text{BitUnpack}(z_i, \eta, \eta)$ ▷ kết quả này có thể nằm ngoài $[-\eta, \eta]$ nếu đầu vào sai định dạng

7: **end for**

8: **for** i **from** 0 **to** $k - 1$ **do**

9: $t_0[i] \leftarrow \text{BitUnpack}(w_i, 2^{d-1} - 1, 2^{d-1})$ ▷ kết quả này luôn nằm trong khoảng đúng

10: **end for**

11: **return** $(\rho, K, tr, s_1, s_2, t_0)$

Tiếp theo, sigEncode và sigDecode chuyển đổi các chữ ký ML-DSA thành các chuỗi byte và ngược lại. Khi xác minh một chữ ký, sigDecode có thể nhận đầu vào xuất phát từ một nguồn không đáng tin cậy. Do đó, cần cẩn trọng khi sử dụng BitUnpack . Như được sử dụng ở đây, BitUnpack luôn trả về các giá trị nằm trong khoảng chính xác.

Thuật toán 26 $\text{sigEncode}(\tilde{c}, \mathbf{z}, \mathbf{h})$ – Mã hóa một chữ ký thành một chuỗi byte

Đầu vào: $\tilde{c} \in \mathbb{B}^{\lambda/4}, \mathbf{z} \in R^\ell$ với các hệ số nằm trong $[-\gamma_1 + 1, \gamma_1], \mathbf{h} \in R_2^k$

Đầu ra: Chữ ký $\sigma \in \mathbb{B}^{\lambda/4 + \ell \cdot 32 \cdot (1 + \text{bitlen}(\gamma_1 - 1)) + \omega + k}$.

```

1:  $\sigma \leftarrow \tilde{c}$ 
2: for  $i$  from 0 to  $\ell - 1$  do
3:    $\sigma \leftarrow \sigma \parallel \text{BitPack}(\mathbf{z}[i], \gamma_1 - 1, \gamma_1)$ 
4: end for
5:  $\sigma \leftarrow \sigma \parallel \text{HintBitPack}(\mathbf{h})$ 
6: return  $\sigma$ 

```

Thuật toán 27 $\text{sigDecode}(\sigma)$ – Thực hiện ngược thủ tục sigEncode

Đầu vào: Chữ ký $\sigma \in \mathbb{B}^{\lambda/4 + \ell \cdot 32 \cdot (1 + \text{bitlen}(\gamma_1 - 1)) + \omega + k}$

Đầu ra: $\tilde{c} \in \mathbb{B}^{\lambda/4}$, $\mathbf{z} \in R^\ell$ với các hệ số nằm trong $[-\gamma_1 + 1, \gamma_1]$, $\mathbf{h} \in R_2^k$ hoặc $\perp$

```

1:  $(\tilde{c}, x_0, \dots, x_{\ell-1}, y) \in \mathbb{B}^{\lambda/4} \times (\mathbb{B}^{32 \cdot (1 + \text{bitlen}(\gamma_1 - 1))})^\ell \times \mathbb{B}^{\omega + k} \leftarrow \sigma$ 
2: for  $i$  from 0 to  $\ell - 1$  do
3:    $\mathbf{z}[i] \leftarrow \text{BitUnpack}(x_i, \gamma_1 - 1, \gamma_1)$   $\triangleright$  kết quả này nằm trong khoảng đúng, vì  $\gamma_1$  là lũy thừa của 2
4: end for
5:  $\mathbf{h} \leftarrow \text{HintBitUnpack}(y)$ 
6: return  $(\tilde{c}, \mathbf{z}, \mathbf{h})$ 

```

w1Encode là một chương trình con cụ thể được sử dụng trong thuật toán [ML-DSA.Sign_internal](#) và trong [ML-DSA.Verify_internal](#). Thủ tục w1Encode mã hóa một vector đa thức $\mathbf{w}_1$ thành một chuỗi byte để nó có thể được xử lý bởi hàm H .

Thuật toán 28 $\text{w1Encode}(\mathbf{w}_1)$ – Mã hóa một vector đa thức $\mathbf{w}_1$ thành một chuỗi byte

Đầu vào: $\mathbf{w}_1 \in R^k$ có các tọa độ đa thức với hệ số nằm trong $[0, (q - 1)/(2\gamma_2) - 1]$

Đầu ra: Một biểu diễn chuỗi byte $\tilde{\mathbf{w}}_1 \in \mathbb{B}^{32k \cdot \text{bitlen}((q-1)/(2\gamma_2)-1)}$

```

1:  $\tilde{\mathbf{w}}_1 \leftarrow ()$ 
2: for  $i$  from 0 to  $k - 1$  do
3:    $\tilde{\mathbf{w}}_1 \leftarrow \tilde{\mathbf{w}}_1 \parallel \text{SimpleBitPack}(\mathbf{w}_1[i], (q - 1)/(2\gamma_2) - 1)$ 
4: end for
5: return  $\tilde{\mathbf{w}}_1$ 

```

9.3. Lấy mẫu giả ngẫu nhiên

Mục này đặc tả các thuật toán khác nhau để tạo giả ngẫu nhiên các đối tượng đại số từ một mầm (seed) ρ , trong đó ρ là một chuỗi byte có độ dài thay đổi tùy thuộc vào thuật toán. Thủ tục đầu tiên được định nghĩa

là SampleInBall. Như trong Mục 4, B_τ ký hiệu tập hợp tất cả các đa thức $c \in R$ sao cho

- Mỗi hệ số của c là $-1, 0$, hoặc 1 , và
- Chính xác τ hệ số của c là khác không.

SampleInBall tạo giả ngẫu nhiên một phần tử của B_τ bằng cách sử dụng hàm XOF của một mầm ρ . Thủ tục này dựa trên thuật toán xáo trộn Fisher-Yates. H được áp dụng cho ρ , và 8 byte đầu tiên của đầu ra được sử dụng để chọn dấu của các phần tử khác không của c^7 . Các byte tiếp theo được sử dụng để chọn vị trí của các phần tử khác không đó.

Thuật toán 29 SampleInBall(ρ) – Lấy mẫu một đa thức $c \in R$ với các hệ số thuộc $\{-1, 0, 1\}$ và trọng số Hamming $\tau \leq 64$

Đầu vào: Một mầm (seed) $\rho \in \mathbb{B}^{\lambda/4}$

Đầu ra: Một đa thức c trong 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:  $h \leftarrow \text{BytesToBits}(s)$  ▷  $h$  là một chuỗi bit có độ dài 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 ▷ lấy mẫu loại bỏ trong  $\{0, \dots, i\}$ 
9:      $(\text{ctx}, j) \leftarrow \text{H.Squeeze}(\text{ctx}, 1)$ 
10:  end while ▷  $j$  là một byte giả ngẫu nhiên mà  $\leq i$ 
11:     $c_i \leftarrow c_j$ 
12:     $c_j \leftarrow (-1)^{h[i+\tau-256]}$ 
13: end for
14: return  $c$ 

```

Các **Thuật toán 30-34** là các thủ tục giả ngẫu nhiên RejNTTPoly, RejBoundedPoly, ExpandA, ExpandS, và ExpandMask. Mỗi thủ tục tạo ra các phần tử của R hoặc T_q dưới các điều kiện đầu vào và đầu ra khác nhau. RejNTTPoly và ExpandA sử dụng hàm XOF G hiệu quả hơn, trong khi ba thủ tục còn lại sử dụng XOF H.

Thủ tục ExpandMask (**Thuật toán 34**) tạo ra một vector đa thức y trong R^ℓ để che giấu khóa bí mật trong thủ tục ML-DSA.Sign_internal (**Thuật toán 7**). Ngoài mầm (seed) ρ , ExpandMask cũng nhận một đầu vào số nguyên μ được kết hợp vào thủ tục giả ngẫu nhiên để tạo ra y .

⁷ Tham số τ luôn nhỏ hơn hoặc bằng 64, do đó 8 byte là đủ để chọn dấu cho tất cả τ thành phần khác không của c .

Thuật toán 30 $\text{RejNTTPoly}(\rho)$ – Lấy mẫu một đa thức $\in T_q$
Đầu vào: Một mầm (seed) $\rho \in \mathbb{B}^{34}$ Đầu ra: Một phần tử $\hat{a} \in T_q$ 1: $j \leftarrow 0$ 2: $\text{ctx} \leftarrow \text{G.Init}()$ 3: $\text{ctx} \leftarrow \text{G.Absorb}(\text{ctx}, \rho)$ 4: while $j < 256$ do 5: $(\text{ctx}, s) \leftarrow \text{G.Squeeze}(\text{ctx}, 3)$ 6: $\hat{a}[j] \leftarrow \text{CoeffFromThreeBytes}(s[0], s[1], s[2])$ 7: if $\hat{a}[j] \neq \perp$ then 8: $j \leftarrow j + 1$ 9: end if 10: end while 11: return $\hat{a}$

Thuật toán 31 $\text{RejBoundedPoly}(\rho)$ – Lấy mẫu một phần tử $a \in R$ với các hệ số nằm trong $[-\eta, \eta]$ được tính toán thông qua lấy mẫu loại bỏ từ ρ
Đầu vào: Một mầm (seed) $\rho \in \mathbb{B}^{66}$ Đầu ra: Một đa thức $a \in R$ 1: $j \leftarrow 0$ 2: $\text{ctx} \leftarrow \text{H.Init}()$ 3: $\text{ctx} \leftarrow \text{H.Absorb}(\text{ctx}, \rho)$ 4: while $j < 256$ do 5: $z \leftarrow \text{H.Squeeze}(\text{ctx}, 1)$ 6: $z_0 \leftarrow \text{CoeffFromHalfByte}(z \bmod 16)$ 7: $z_1 \leftarrow \text{CoeffFromHalfByte}(\lfloor z/16 \rfloor)$ 8: if $z_0 \neq \perp$ then 9: $a_j \leftarrow z_0$ 10: $j \leftarrow j + 1$ 11: end if 12: if $z_1 \neq \perp$ and $j < 256$ then 13: $a_j \leftarrow z_1$ 14: $j \leftarrow j + 1$

```

15:   end if
16: end while
17: return  $a$ 

```

Thuật toán 32 $\text{ExpandA}(\rho)$ – Lấy mẫu một ma trận $\hat{A}$ kích thước $k \times \ell$ gồm các phần tử của T_q

Đầu vào: Một mầm (seed) $\rho \in \mathbb{B}^{32}$

Đầu ra: Ma trận $\hat{A} \in (T_q)^{k \times \ell}$

```

1: for  $r$  from 0 to  $k - 1$  do
2:   for  $s$  from 0 to  $\ell - 1$  do
3:      $\rho' \leftarrow \rho \parallel \text{IntegerToBytes}(s, 1) \parallel \text{IntegerToBytes}(r, 1)$ 
4:      $\hat{A}[r, s] \leftarrow \text{RejNTTPoly}(\rho')$  ▷ mầm  $\rho'$  phụ thuộc vào  $s$  và  $r$ 
5:   end for
6: end for
7: return  $\hat{A}$ 

```

Thuật toán 33 $\text{ExpandS}(\rho)$ – Lấy mẫu các vector $s_1 \in R^\ell$ và $s_2 \in R^k$, mỗi vector có các tọa độ đa thức với hệ số nằm trong đoạn $[-\eta, \eta]$

Đầu vào: Một mầm (seed) $\rho \in \mathbb{B}^{64}$

Đầu ra: Các vector s_1, s_2 chứa các đa thức trong R

```

1: for  $r$  from 0 to  $\ell - 1$  do
2:    $s_1[r] \leftarrow \text{RejBoundedPoly}(\rho \parallel \text{IntegerToBytes}(r, 2))$  ▷ mầm phụ thuộc vào  $r$ 
3: end for
4: for  $r$  from 0 to  $k - 1$  do
5:    $s_2[r] \leftarrow \text{RejBoundedPoly}(\rho \parallel \text{IntegerToBytes}(r + \ell, 2))$  ▷ mầm phụ thuộc vào  $r + \ell$ 
6: end for
7: return  $(s_1, s_2)$ 

```

Thuật toán 34 $\text{ExpandMask}(\rho, \mu)$ – Lấy mẫu vector $y \in R^\ell$ sao cho mỗi đa thức $y[r]$ có các hệ số nằm trong khoảng từ $-\gamma_1 + 1$ đến γ_1

Đầu vào: Một mầm (seed) $\rho \in \mathbb{B}^{64}$ và một số nguyên không âm μ

Đầu ra: Vector $y \in R^\ell$

```

1:  $c \leftarrow 1 + \text{bitlen}(\gamma_1 - 1)$   $\triangleright \gamma_1$  luôn là một lũy thừa của 2
2: for  $r$  from 0 to  $\ell - 1$  do
3:    $\rho' \leftarrow \rho \parallel \text{IntegerToBytes}(\mu + r, 2)$ 
4:    $v \leftarrow H(\rho', 32c)$   $\triangleright$  mầm phụ thuộc vào  $\mu + r$ 
5:    $y[r] \leftarrow \text{BitUnpack}(v, \gamma_1 - 1, \gamma_1)$ 
6: end for
7: return  $y$ 

```

9.4. Các bit bậc cao, bậc thấp và gợi ý

Đặc tả này sử dụng các hàm hỗ trợ Power2Round, Decompose, HighBits, LowBits, MakeHint, và UseHint và định nghĩa tường minh các hàm này, trong đó $r \in \mathbb{Z}_q, r_1, r_0 \in \mathbb{Z}$, và h là một giá trị Boolean (hay tương đương là một phần tử thuộc $\mathbb{Z}_2$). Tuy nhiên, đặc tả này cũng sử dụng các hàm này trong trường hợp $\mathbf{r}, \mathbf{z} \in R_q^k, \mathbf{r}_1, \mathbf{r}_0 \in R^k$, và $\mathbf{h} \in R_2^k$. Trong trường hợp này, các hàm được áp dụng cho từng hệ số của các đa thức trong vec tơ. Cụ thể:

Với $\mathbf{r} \in R_q^k$, định nghĩa $(\mathbf{r}_1, \mathbf{r}_0) \in (R^k)^2 = \text{Power2Round}(\mathbf{r})$ sao cho:

$$((\mathbf{r}_1[i])_j, (\mathbf{r}_0[i])_j) = \text{Power2Round}((\mathbf{r}[i])_j).$$

- Với $\mathbf{r} \in R_q^k$, định nghĩa $(\mathbf{r}_1, \mathbf{r}_0) \in (R^k)^2 = \text{Decompose}(\mathbf{r})$ sao cho:

$$((\mathbf{r}_1[i])_j, (\mathbf{r}_0[i])_j) = \text{Decompose}((\mathbf{r}[i])_j).$$

- Với $\mathbf{r} \in R_q^k$, định nghĩa $\mathbf{r}_1 = \text{HighBits}(\mathbf{r})$ sao cho:

$$((\mathbf{r}_1[i])_j) = \text{HighBits}((\mathbf{r}[i])_j)$$

- Với $\mathbf{r} \in R_q^k$, định nghĩa $\mathbf{r}_0 = \text{LowBits}(\mathbf{r})$ sao cho:

$$(\mathbf{r}_0[i])_j = \text{LowBits}((\mathbf{r}[i])_j).$$

- Với $\mathbf{z}, \mathbf{r} \in R_q^k$, định nghĩa $\mathbf{h} \in R_2^k = \text{MakeHint}(\mathbf{z}, \mathbf{r})$ sao cho:

$$(\mathbf{h}[i])_j = \text{MakeHint}((\mathbf{z}[i])_j, (\mathbf{r}[i])_j)$$

- Với $\mathbf{h} \in R_2^k$ và $\mathbf{r} \in R_q^k$, định nghĩa $\mathbf{r}_1 \in R^k = \text{UseHint}(\mathbf{h}, \mathbf{r})$ sao cho:

$$\mathbf{r}_1[i]_j = \text{UseHint}((\mathbf{h}[i])_j, (\mathbf{r}[i])_j)$$

Các thuật toán này được sử dụng để hỗ trợ việc tối ưu hóa nén khóa của ML-DSA. Chúng bao gồm việc loại bỏ d bit bậc thấp của mỗi hệ số thuộc vector đa thức t từ khóa công khai bằng cách sử dụng hàm Power2Round. Tuy nhiên, để việc tối ưu hóa này hoạt động, một thông tin bổ sung gọi là "gợi ý" cần được cung cấp trong chữ ký để cho phép bên xác minh khôi phục đủ lượng thông tin trong các bit khóa công khai đã bị loại bỏ nhằm xác minh chữ ký. Các gợi ý được tạo ra trong quá trình ký và được sử dụng trong quá trình xác minh tương ứng bởi các hàm MakeHint và UseHint. Trong việc xác minh một chữ ký hợp lệ, gợi ý cho phép bên xác minh khôi phục $\mathbf{w}_1 \in R^k$, đại diện cho $\mathbf{w} \in R_q^k$ được làm tròn đến một bội số gần nhất của $\alpha = 2\gamma_2$. Người ký trực tiếp thu được $\mathbf{w}_1$ bằng cách sử dụng hàm HighBits, và phần bị làm tròn đi (tức là $\mathbf{r}_0$) thu được bởi LowBits. $\mathbf{r}_0$ được người ký sử dụng trong quy trình lấy mẫu loại bỏ.

Power2Round phân rã một đầu vào $r \in \mathbb{Z}_q$ thành các số nguyên đại diện cho các bit bậc cao và bậc thấp của $r \bmod q$ theo cách xử lý bit trực tiếp, $r \bmod q = r_1 \cdot 2^d + r_0$, trong đó $r_0 = (r \bmod q) \bmod \pm 2^d$ và $r_1 = (r \bmod q - r_0)/2^d$.

Tuy nhiên, vì mục đích của các tính toán liên quan đến gợi ý, phương pháp phân rã r này có một đặc tính không mong muốn là khi r gần với $q - 1$ hoặc 0, một sai số làm tròn nhỏ trong r có thể khiến r_1 thay đổi nhiều hơn 1, ngay cả khi đã tính đến sự quay vòng. Trái ngược với các cặp giá trị không bằng nhau khác của $r_1 \cdot 2^d$ và $r'_1 \cdot 2^d$, khoảng cách $(\bmod q)$ giữa $\lfloor q/2^d \rfloor \cdot 2^d$ và 0 có thể rất nhỏ.

Để tránh vấn đề này, đặc tả này định nghĩa hàm Decompose, tương tự như Power2Round ngoại trừ:

- r thường được phân rã thành $r \bmod q = r_1 \cdot \alpha + r_0$, trong đó $\alpha = 2\gamma_2$ là một ước của $q - 1$.
- Nếu quy trình làm tròn trực tiếp trả về $(r_1 = (q - 1)/\alpha, r_0 \in [-(\alpha/2) + 1, \alpha/2])$, Decompose thay vào đó sẽ trả về $(r_1 = 0, r_0 - 1)$.

Các hàm HighBits và LowBits - tương ứng chỉ trả về r_1 và r_0 - và MakeHint cùng UseHint đều sử dụng Decompose. Để thảo luận thêm về các đặc tính toán học của những hàm này liên quan đến tính đúng đắn và độ an toàn của ML-DSA, xem Mục 2.4 trong [5].

Thuật toán 35 Power2Round(r) – Phân rã r thành (r_1, r_0) sao cho $r \equiv r_1 2^d + r_0 \bmod q$

Đầu vào: $r \in \mathbb{Z}_q$

Đầu ra: Các số nguyên (r_1, r_0)

1: $r^+ \leftarrow r \bmod q$

2: $r_0 \leftarrow r^+ \bmod \pm 2^d$

3: **return** $((r^+ - r_0)/2^d, r_0)$

Thuật toán 36 Decompose(r) – Phân rã r thành (r_1, r_0) sao cho $r \equiv r_1(2\gamma_2) + r_0 \bmod q$

Đầu vào: $r \in \mathbb{Z}_q$

Đầu ra: Các số nguyên (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)

Thuật toán 37 HighBits(r) – Trả về r_1 từ đầu ra của Decompose(r)

Đầu vào: $r \in \mathbb{Z}_q$
Đầu ra: Số nguyên r_1
1: $(r_1, r_0) \leftarrow \text{Decompose}(r)$
2: **return** r_1

Thuật toán 38 LowBits(r) – Trả về r_0 từ đầu ra của Decompose(r)

Đầu vào: $r \in \mathbb{Z}_q$
Đầu ra: Số nguyên r_0
1: $(r_1, r_0) \leftarrow \text{Decompose}(r)$
2: **return** r_0

Thuật toán 39 MakeHint(z, r) – Tính toán bit gợi ý chỉ ra liệu việc cộng z vào r có làm thay đổi các bit cao của r hay không

Đầu vào: $z, r \in \mathbb{Z}_q$
Đầu ra: Giá trị Boolean
1: $r_1 \leftarrow \text{HighBits}(r)$
2: $v_1 \leftarrow \text{HighBits}(r + z)$
3: **return** $\llbracket r_1 \neq v_1 \rrbracket$

Thuật toán 40 UseHint(h, r) – Trả về các bit cao của r được điều chỉnh theo gợi ý h

Đầu vào: Boolean $h, r \in \mathbb{Z}_q$
Đầu ra: $r_1 \in \mathbb{Z}$ với $0 \leq r_1 \leq \frac{q-1}{2\gamma_2-1}$
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

9.5. NTT và NTT^{-1}

Các thuật toán sau đây cài đặt NTT và phép nghịch đảo của nó (NTT^{-1}), điều này rất quan trọng đối với tính hiệu quả. Có những tối ưu hóa khác không được bao gồm trong tiêu chuẩn này. Cụ thể, $\text{mod } q$ và $\text{mod}^{\pm} q$ là các phép toán tốn kém mà việc sử dụng chúng có thể được giảm thiểu bằng cách sử dụng Phép nhân Montgomery.

Một phần tử của R_q là một đa thức trong $\mathbb{Z}_q[X]/(X^{256} + 1)$, và một phần tử của T_q là một bộ trong $\prod_{j=0}^{255} \mathbb{Z}_q$. Thuật toán NTT nhận một đa thức $w \in R_q$ làm đầu vào và trả về $\hat{w} \in T_q$. NTT^{-1} nhận $\hat{w} \in T_q$ làm đầu vào và trả về w sao cho $\hat{w} = \text{NTT}(w)$.

Tiêu chuẩn này luôn phân biệt giữa các phần tử của R_q và các phần tử của T_q . Tuy nhiên, cấu trúc dữ liệu tự nhiên cho cả hai tập hợp này là một mảng số nguyên có kích thước 256. Điều này sẽ cho phép các thuật toán NTT và NTT^{-1} thực hiện tính toán tại chỗ trên một mảng số nguyên được truyền bằng tham chiếu. Tối ưu hóa đó không được bao gồm trong tiêu chuẩn này.

Trong Mục 4.1, $\zeta = 1753 \in \mathbb{Z}_q$, là căn bậc 512 của đơn vị modulo q . Với đầu vào $w \in R_q$, thuật toán đưa ra

$$\text{NTT}(w) = (w(\zeta_0), w(\zeta_1), \dots, w(\zeta_{255})) \in T_q \quad (7.1)$$

trong đó, $\zeta_i = \zeta^{2^{\text{BitRev}_8(i)+1}} \text{mod } q$.

Các giá trị $\zeta^{\text{BitRev}_8(k)} \text{mod } q$ với $k = 1, \dots, 255$ được sử dụng ở dòng 10 của **Thuật toán 41** và **Thuật toán 42** được tính toán trước thành một mảng zetas[1..255]. Bảng hệ số zeta được trình bày trong **Phụ lục B**. Nếu sử dụng phép nhân Montgomery (xem **Phụ lục A**), thì mảng zeta thường được lưu trữ dưới dạng Montgomery.

NTT và NTT^{-1} sử dụng BitRev_8 , hàm này đảo ngược thứ tự các bit trong một số nguyên 8-bit.

Thuật toán 41 $\text{NTT}(w)$ – Tính toán phép biến đổi NTT

Đầu vào: Đa thức $w(X) = \sum_{j=0}^{255} w_j X^j \in R_q$

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

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:	$z \leftarrow \text{zetas}[m]$	$\triangleright z \leftarrow \zeta^{\text{BitRev}_8(m)} \bmod q$
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 $\hat{w}$	

Thuật toán 42 $\text{NTT}^{-1}(\hat{w})$ – Tính toán nghịch đảo của phép biến đổi NTT		
Đầu vào: $\hat{w} = (\hat{w}[0], \dots, \hat{w}[255]) \in T_q$		
Đầu ra: Đa thức $w(X) = \sum_{j=0}^{255} w_j X^j \in R_q$		
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:	$z \leftarrow -\text{zetas}[m]$	$\triangleright z \leftarrow -\zeta^{\text{BitRev}_8(m)} \bmod q$
11:	for j from $start$ to $start + len - 1$ do	
12:	$t \leftarrow w_j$	
13:	$w_j \leftarrow (t + w_{j+len}) \bmod q$	
14:	$w_{j+len} \leftarrow (t - w_{j+len}) \bmod q$	
15:	$w_{j+len} \leftarrow (z \cdot w_{j+len}) \bmod q$	
16:	end for	
17:	$start \leftarrow start + 2 \cdot len$	

```

18:  end while
19:  len  $\leftarrow 2 \cdot len$ 
20:  end while
21:   $f \leftarrow 8347681$   $\triangleright f = 256^{-1} \bmod q$ 
22:  for  $j$  from 0 to 255 do
23:     $w_j \leftarrow (f \cdot w_j) \bmod q$ 
24:  end for
25:  return  $w$ 

```

Thuật toán 43 $\text{BitRev}_8(r)$ – *Biến đổi một byte bằng cách đảo ngược thứ tự các bit trong biểu diễn nhị phân 8 bit của nó*

Đầu vào: Một byte $m \in [0, 255]$.

Đầu ra: Một byte $r \in [0, 255]$

```

1:  $b \leftarrow \text{IntegerToBits}(m, 8)$ 
2:  $b_{\text{rev}} \in \{0, 1\}^8 \leftarrow (0, \dots, 0)$ 
3: for  $i$  from 0 to 7 do
4:    $b_{\text{rev}}[i] \leftarrow b[7 - i]$ 
5: end for
6:  $r \leftarrow \text{BitsToInteger}(b_{\text{rev}}, 8)$ 
7: return  $r$ 

```

9.6. Số học dưới phép biến đổi NTT

Phép biến đổi NTT chuyển đổi các phần tử của vành R_q (trong đó phép cộng và phép nhân tương ứng được ký hiệu là $+$ và $\cdot$) thành các phần tử của vành T_q (trong đó phép cộng và phép nhân tương ứng được ký hiệu là $+$ và $\circ$). Phần này đưa ra các thuật toán tường minh cho đại số tuyến tính trên vành T_q .

Vành T_q được định nghĩa là vành tích trực tiếp $\prod_{i=0}^{255} \mathbb{Z}_q$. Do đó, một phần tử $\hat{a} \in T_q$ là một mảng có độ dài 256, và các phần tử của nó được ký hiệu là $\hat{a}[0], \hat{a}[1], \dots, \hat{a}[255] \in \mathbb{Z}_q$.

Thuật toán 44 $\text{AddNTT}(\hat{a}, \hat{b})$ – *Tính tổng $\hat{a} + \hat{b}$ của hai phần tử $\hat{a}, \hat{b} \in T_q$*

Đầu vào: $\hat{a}, \hat{b} \in T_q$

Đầu ra: $\hat{c} \in T_q$

```

1: for  $i$  from 0 to 255 do
2:    $\hat{c}[i] \leftarrow \hat{a}[i] + \hat{b}[i]$ 

```

```

3: end for
4: return  $\hat{c}$ 

```

Thuật toán 45 $\text{MultiplyNTT}(\hat{a}, \hat{b})$ – Tính tích $\hat{a} \circ \hat{b}$ của hai phần tử $\hat{a}, \hat{b} \in T_q$

Đầu vào: $\hat{a}, \hat{b} \in T_q$

Đầu ra: $\hat{c} \in T_q$

```

1: for  $i$  from 0 to 255 do
2:    $\hat{c}[i] \leftarrow \hat{a}[i] \cdot \hat{b}[i]$ 
3: end for
4: return  $\hat{c}$ 

```

Thuật toán 46 $\text{AddVectorNTT}(\hat{v}, \hat{w})$ – Tính tổng $\hat{v} + \hat{w}$ của hai vector $\hat{v}, \hat{w}$ trên T_q

Đầu vào: $\ell \in \mathbb{N}, \hat{v} \in T_q^\ell, \hat{w} \in T_q^\ell$

Đầu ra: $\hat{u} \in T_q^\ell$

```

1: for  $i$  from 0 to  $\ell - 1$  do
2:    $\hat{u}[i] \leftarrow \text{AddNTT}(\hat{v}[i], \hat{w}[i])$ 
3: end for
4: return  $\hat{u}$ 

```

Thuật toán 47 $\text{ScalarVectorNTT}(\hat{c}, \hat{v})$ – Tính tích $\hat{c} \circ \hat{v}$ của một vô hướng $\hat{c}$ và một vector $\hat{v}$ trên T_q

Đầu vào: $\hat{c} \in T_q, \ell \in \mathbb{N}, \hat{v} \in T_q^\ell$

Đầu ra: $\hat{w} \in T_q^\ell$

```

1: for  $i$  from 0 to  $\ell - 1$  do
2:    $\hat{w}[i] \leftarrow \text{MultiplyNTT}(\hat{c}, \hat{v}[i])$ 
3: end for
4: return  $\hat{w}$ 

```

Thuật toán 48 $\text{MatrixVectorNTT}(\hat{M}, \hat{v})$ – Tính tích $\hat{M} \circ \hat{v}$ của một ma trận $\hat{M}$ và một vector $\hat{v}$ trên T_q

Đầu vào: $k, \ell \in \mathbb{N}, \hat{M} \in T_q^{k \times \ell}, \hat{v} \in T_q^\ell$

Đầu ra: $\hat{w} \in T_q^k$

```

1:  $\hat{w} \leftarrow 0^k$ 

```

```

2: for  $i$  from 0 to  $k - 1$  do
3:   for  $j$  from 0 to  $\ell - 1$  do
4:      $\hat{w}[i] \leftarrow \text{AddNTT}(\hat{w}[i], \text{MultiplyNTT}(\hat{M}[i, j], \hat{v}[j]))$ 
5:   end for
6: end for
7: return  $\hat{w}$ 

```

PHỤ LỤC A – Phép nhân Montgomery

Tiêu chuẩn này sử dụng các phép nhân môđun có dạng $a \cdot b$ theo môđun q . Đây là một phép toán tốn kém, thường được tăng tốc trong thực tế bằng cách sử dụng phép nhân Montgomery.

Nếu a là một số nguyên theo môđun q , thì dạng Montgomery của nó với thừa số nhân 2^{32} là $r \equiv a \cdot 2^{32} \bmod q^8$.

Giả sử hai số nguyên u và v theo môđun q ở dạng Montgomery. Tích của chúng theo môđun q là $c = u \cdot v \cdot 2^{-32}$, cũng ở dạng Montgomery. Nếu tích số nguyên của u và v không gây tràn số nguyên có dấu 64 bit, thì có thể tính c bằng cách trước tiên thực hiện phép nhân số nguyên $u \cdot v$, rồi “rút gọn” tích bằng cách nhân với 2^{-32} theo môđun q . Phép toán cuối cùng này có thể được thực hiện hiệu quả như sau.

Hàm MontgomeryReduce nhận đầu vào là một số nguyên a có giá trị tuyệt đối không vượt quá $2^{31}q$. Hàm trả về một số nguyên r sao cho $r = a \cdot 2^{-32} \bmod q$. Đầu ra ở dạng Montgomery với thừa số nhân $2^{32} \bmod q$. Một bản cài đặt thường nhận đầu vào 64 bit và trả về đầu ra 32 bit. Phép toán “môđun 2^{32} ” chỉ đơn giản trích xuất 32 bit thấp nhất của một giá trị 64 bit. Giá trị $(a - t \cdot q)$ tại dòng 3 là một số nguyên chia hết cho 2^{32} . Do đó, phép chia chỉ đơn giản là lấy 32 bit cao nhất của một giá trị 64 bit. Việc trích xuất bốn byte bậc thấp hoặc bậc cao thường được thực hiện bằng cách ép kiểu.

Thuật toán 49 MontgomeryReduce(a) – Tính $a \cdot 2^{-32} \bmod q$

Đầu vào: Số nguyên a với $-2^{31}q \leq a \leq 2^{31}q$

Đầu ra: $r \equiv a \cdot 2^{-32} \bmod q$

```

1:  $QINV \leftarrow 58728449$  ▷ nghịch đảo của  $q$  theo modulo  $2^{32}$ 
2:  $t \leftarrow ((a \bmod 2^{32}) \cdot QINV) \bmod 2^{32}$ 
3:  $r \leftarrow (a - t \cdot q) / 2^{32}$ 
4: return  $r$ 

```

Với thuật toán này, tích môđun của a và b là $c = \text{MontgomeryReduce}(a \cdot b)$, trong đó a, b và c đều ở dạng Montgomery. Giá trị trả về của thuật toán không nhất thiết có giá trị tuyệt đối nhỏ hơn q , nhưng có giá trị

⁸ Mục này không phân biệt các phiên bản khác nhau của toán tử “mod”. Có ba phiên bản như vậy của “ $x = a$ môđun q ”: i) $x \in [0, q - 1]$; ii) $x \in [-q/2, q/2]$; iii) $x \in [-q + 1, q - 1]$. Phiên bản cuối cùng tương ứng với toán tử “%” trong hầu hết các ngôn ngữ lập trình.

tuyệt đối nhỏ hơn $2q$. Điều này không đáng lo ngại trong thực tế vì mục tiêu của phép nhân Montgomery là thực hiện hiệu quả các phép toán với các giá trị môđun vừa trong một thanh ghi 32 bit. Nếu cần, kết quả có thể được chuẩn hóa thành một số nguyên trong $(-q, q)$ bằng một phép so sánh và một phép cộng số nguyên.

Việc chuyển đổi một số nguyên theo môđun q sang dạng Montgomery bằng cách nhân với 2^{32} theo môđun q là một phép toán tốn kém. Khi cần thực hiện một dãy phép toán môđun, các toán hạng được chuyển sang dạng Montgomery một lần. Sau đó, các phép toán được thực hiện và thừa số 2^{32} được loại bỏ khỏi kết quả cuối cùng.

PHỤ LỤC B – Bảng Zetas

Các giá trị $\zeta^{\text{BitRev}_8(k)} \bmod q$ với $k = 1, \dots, 255$, được sử dụng trong các Thuật toán NTT 41 và 42, có thể được tính trước và lưu trữ trong một mảng zetas[1..255]. Bảng zetas này được cho dưới đây.

zetas[0..255] = {

0,	4808194,	3765607,	3761513,	5178923,	5496691,	5234739,	5178987,
7778734,	3542485,	2682288,	2129892,	3764867,	7375178,	557458,	7159240,
5010068,	4317364,	2663378,	6705802,	4855975,	7946292,	676590,	7044481,
5152541,	1714295,	2453983,	1460718,	7737789,	4795319,	2815639,	2283733,
3602218,	3182878,	2740543,	4793971,	5269599,	2101410,	3704823,	1159875,
394148,	928749,	1095468,	4874037,	2071829,	4361428,	3241972,	2156050,
3415069,	1759347,	7562881,	4805951,	3756790,	6444618,	6663429,	4430364,
5483103,	3192354,	556856,	3870317,	2917338,	1853806,	3345963,	1858416,
3073009,	1277625,	5744944,	3852015,	4183372,	5157610,	5258977,	8106357,
2508980,	2028118,	1937570,	4564692,	2811291,	5396636,	7270901,	4158088,
1528066,	482649,	1148858,	5418153,	7814814,	169688,	2462444,	5046034,
4213992,	4892034,	1987814,	5183169,	1736313,	235407,	5130263,	3258457,
5801164,	1787943,	5989328,	6125690,	3482206,	4197502,	7080401,	6018354,
7062739,	2461387,	3035980,	621164,	3901472,	7153756,	2925816,	3374250,
1356448,	5604662,	2683270,	5601629,	4912752,	2312838,	7727142,	7921254,
348812,	8052569,	1011223,	6026202,	4561790,	6458164,	6143691,	1744507,
1753,	6444997,	5720892,	6924527,	2660408,	6600190,	8321269,	2772600,
1182243,	87208,	636927,	4415111,	4423672,	6084020,	5095502,	4663471,
8352605,	822541,	1009365,	5926272,	6400920,	1596822,	4423473,	4620952,
6695264,	4969849,	2678278,	4611469,	4829411,	635956,	8129971,	5925040,
4234153,	6607829,	2192938,	6653329,	2387513,	4768667,	8111961,	5199961,
3747250,	2296099,	1239911,	4541938,	3195676,	2642980,	1254190,	8368000,
2998219,	141835,	8291116,	2513018,	7025525,	613238,	7070156,	6161950,
7921677,	6458423,	4040196,	4908348,	2039144,	6500539,	7561656,	6201452,
6757063,	2105286,	6006015,	6346610,	586241,	7200804,	527981,	5637006,
6903432,	1994046,	2491325,	6987258,	507927,	7192532,	7655613,	6545891,
5346675,	8041997,	2647994,	3009748,	5767564,	4148469,	749577,	4357667,

3980599, 2569011, 6764887, 1723229, 1665318, 2028038, 1163598, 5011144,
3994671, 8368538, 7009900, 3020393, 3363542, 214880, 545376, 7609976,
3105558, 7277073, 508145, 7826699, 860144, 3430436, 140244, 6866265,
6195333, 3123762, 2358373, 6187330, 5365997, 6663603, 2926054, 7987710,
8077412, 3531229, 4405932, 4606686, 1900052, 7598542, 1054478, 7648983 }

PHỤ LỤC C — Các giới hạn vòng lặp

Có bốn thuật toán trong tiêu chuẩn này có các vòng lặp với số lần lặp không xác định, mặc dù trong mỗi trường hợp, số lần lặp kỳ vọng là một hằng số nhỏ. Ba trong số bốn thuật toán thực hiện lấy mẫu từ đầu ra của một XOF, trong đó lượng đầu ra cần lấy từ XOF tỷ lệ thuận với số lần lặp được thực hiện.

Các bản cài đặt không nên giới hạn số lần lặp trong các vòng lặp này hoặc lượng đầu ra được trích xuất từ các XOF khi thực hiện các hàm này⁹. Nếu một bản cài đặt giới hạn số lần lặp hoặc số byte có thể được trích xuất từ XOF, thì không được sử dụng giới hạn thấp hơn các giá trị được trình bày trong Bảng 3. Các giới hạn này cho xác suất chạm giới hạn xấp xỉ 2⁻²⁵⁶ (hoặc nhỏ hơn) trong một bản cài đặt đúng tiêu chuẩn này. Xác suất được tính theo các giả thiết tiêu chuẩn về phân phối đầu ra của các XOF và các hàm băm.

Bảng 3. Các giới hạn vòng lặp while và đầu ra XOF để xác suất thất bại không vượt quá 2⁻²⁵⁶

Thuật toán	Giới hạn tối thiểu cho phép (số lần lặp)	Giới hạn tối thiểu cho phép (số byte đầu ra XOF)
ML-DSA.Sign_internal	821	Không áp dụng
RejBoundedPoly	481	481
RejNTPoly	298	894
SampleInBall	121	221

Các bản cài đặt có thể giới hạn số lần lặp của một vòng lặp while hoặc số byte lấy từ XOF để không vượt quá các giá trị tối đa trong Bảng 3. Nếu sử dụng tùy chọn này và vượt quá số lần lặp hoặc số byte đầu ra XOF tối đa, thuật toán phải hủy tất cả các kết quả trung gian. Nếu tạo ra một giá trị trả về hoặc một ngoại lệ, thì giá trị trả về hoặc ngoại lệ đó phải giống nhau đối với mọi lần thực hiện vượt quá số lần lặp hoặc số byte đầu ra tối đa.

Về cơ bản, việc sử dụng một giới hạn lớn hơn mức cần thiết không làm giảm hiệu năng, vì giới hạn chỉ bị chạm tới khi vòng lặp được thực hiện có lỗi. Vì lý do này, các giới hạn đã được lựa chọn sao cho xác suất chạm tới chúng có cận dưới là 2⁻²⁵⁶.

ML-DSA.Sign_internal

⁹ RejBoundedPoly, RejNTPoly và SampleInBall sử dụng các API gia tăng được mô tả trong Mục 3.7 để trích xuất lượng đầu ra cần thiết từ các XOF, vì lượng cần thiết không được biết trước

54

Bảng 1 chứa số lần lặp kỳ vọng trong vòng lặp lấy mẫu loại bỏ của ML-DSA.Sign_internal. Các giá trị này lần lượt là 4.36, 5.14 và 3.91 đối với các cấp an toàn 2, 3 và 5. Do đó, xác suất để số lần lặp vượt quá n nhỏ hơn hoặc bằng $\left(\frac{5.1-1}{5.1}\right)^n$ đối với tất cả các cấp an toàn. Giải bất đẳng thức $\left(\frac{5.1-1}{5.1}\right)^n \leq 2^{-256}$ thu được $n = 814$.

RejBoundedPoly

Gọi X là số hệ số được sinh ra trong n lần lặp của vòng lặp while trong RejBoundedPoly¹⁰. Khi đó, X có phân phối $Binomial(2n, \theta)$, trong đó θ bằng $\frac{9}{16}$ hoặc $\frac{15}{16}$, tùy thuộc vào tham số η . Với $\theta = \frac{9}{16}$, xác suất sinh được ít hơn 256 hệ số trong 481 lần lặp của vòng lặp chính trong RejBoundedPoly nhỏ hơn 2^{-256} . Mỗi lần lặp tiêu thụ một byte đầu ra từ H

RejNTTPoly

Số hệ số hợp lệ được sinh ra trong n lần gọi G.Squeeze trong RejNTTPoly có phân phối $Binomial(n, 2^{-23}q)$. Suy ra rằng sau 298 lần gọi, xác suất thất bại nhỏ hơn 2^{-256} . Mỗi lần lặp tiêu thụ ba byte đầu ra từ G.

SampleInBall

Bước 9 trong SampleInBall được thực hiện mỗi khi một byte giả ngẫu nhiên lớn hơn một giá trị i trong khoảng $[256 - \tau, 255]$. Tham số τ lần lượt là 39, 49 và 60 đối với các cấp an toàn 2, 3 và 5. Do đó, xác suất để bước này được thực hiện nhiều hơn n lần trong một lần lặp của vòng lặp for nhỏ hơn hoặc bằng $\left(\frac{59}{256}\right)^n \leq \left(\frac{\tau}{256}\right)^n$. Giải bất đẳng thức $\left(\frac{59}{256}\right)^n \leq 2^{-256}$ thu được cận $n = 121$ cho vòng lặp while tại bước 8 của SampleInBall. Mỗi lần lặp tiêu thụ một byte đầu ra từ H.

Mỗi lần gọi SampleInBall trích xuất tám byte từ H, sau đó thực hiện τ lần lặp của vòng lặp for, mỗi lần trích xuất một lượng dữ liệu không xác định từ H. Xác suất cần nhiều hơn n byte đầu ra từ H trong một lần thực hiện SampleInBall với một giá trị τ cho trước là

$$P(n, \tau) = \begin{cases} 1 & \text{nếu } n \leq 8 \\ 0 & \text{nếu } \tau = 1 \text{ và } n > 8 \\ \left(\frac{257 - \tau}{256}\right)P(n - 1, \tau - 1) + \left(\frac{\tau - 1}{256}\right)P(n - 1, \tau) & \text{nếu } \tau > 1 \text{ và } n > 8 \end{cases}$$

$P(n, 60)$ nhỏ hơn 2^{-256} khi n bằng 221 hoặc lớn hơn. Các bản triển khai có thể giới hạn số byte được trích xuất từ H ở mức $n \geq 221$. Những bản triển khai như vậy phải dừng việc thực hiện SampleInBall, trả về một hằng số biểu thị lỗi và không có đầu ra nào khác, đồng thời hủy tất cả các kết quả trung gian sau khi đã tiêu thụ n byte đầu ra.

¹⁰ Lưu ý rằng mỗi lần lặp sinh ra 0, 1 hoặc 2 hệ số.

PHỤ LỤC D — Những khác biệt so với bản hồ sơ đề trình CRYSTALS-Dilithium

ML-DSA được phát triển từ bản đề trình Phiên bản 3.1 của CRYSTALS-DILITHIUM [6]. Phiên bản 3.1 khác đôi chút so với phiên bản gần đây nhất xuất hiện trên trang web của NIST (tức là Phiên bản 3 của CRYSTALS-DILITHIUM [5]). Các mục D.1, D.2 và D.3 lần lượt ghi lại những khác biệt giữa Phiên bản 3 và 3.1, những khác biệt giữa Phiên bản 3.1 và dự thảo công khai đầu tiên của ML-DSA, cùng những khác biệt giữa dự thảo công khai đầu tiên và tiêu chuẩn ML-DSA được công bố trong tiêu chuẩn này.

D.1 Những khác biệt giữa Phiên bản 3.1 và phiên bản Vòng 3 của CRYSTALS-Dilithium

Độ dài của các biến ρ' (mẫu ngẫu nhiên bí mật) và μ (đại diện của thông điệp) trong thuật toán ký được tăng từ 384 lên 512 bit. Việc tăng độ dài của μ khắc phục một điểm yếu về an toàn xuất hiện trong hồ sơ đề trình vòng ba, trong đó một tấn công va chạm đối với SHAKE256 có đầu ra 384 bit sẽ khiến các tham số hướng tới cấp an toàn 5 của NIST chỉ có thể đáp ứng cấp an toàn 4 [32].

Ngoài ra, độ dài của biến tr (giá trị băm của khóa công khai) được giảm từ 384 xuống 256 bit. Trong quá trình sinh khóa, biến ς được đổi tên thành ρ' và tăng kích thước từ 256 bit lên 512 bit.

D.2 Những khác biệt giữa Phiên bản 3.1 của CRYSTALS-DILITHIUM và dự thảo công khai đầu tiên của FIPS 204

Để bảo đảm các tính chất được nêu trong [14], ML-DSA tăng độ dài của tr lên 512 bit và tăng độ dài của $\tilde{c}$ lên lần lượt 384 và 512 bit đối với các bộ tham số ML-DSA-65 và ML-DSA-87. Trong dự thảo ML-DSA, chỉ 256 bit đầu tiên của $\tilde{c}$ được sử dụng để sinh c .

Trong Phiên bản 3.1 của hồ sơ đề trình CRYSTALS-DILITHIUM, phiên bản mặc định của thuật toán ký có tính tất định, với ρ' được sinh giả ngẫu nhiên từ khóa bí mật của bên ký và thông điệp; một phiên bản tùy chọn của thuật toán ký thay vào đó lấy mẫu ρ' như một chuỗi ngẫu nhiên 512 bit. Trong ML-DSA, ρ' được sinh bằng một thủ tục “có dự phòng”, trong đó ρ' được suy ra một cách giả ngẫu nhiên từ khóa bí mật của bên ký, thông điệp và một chuỗi 256 bit rnd ; theo mặc định, chuỗi này nên được sinh bởi một bộ sinh bit ngẫu nhiên (RBG) được quy định trong tiêu chuẩn này. Tiêu chuẩn ML-DSA cũng cho phép một phiên bản tất định tùy chọn, trong đó rnd là một chuỗi hằng 256 bit.

Dự thảo tiêu chuẩn ML-DSA cũng chứa giả mã vô tình bỏ sót một phép kiểm tra đầu vào sai định dạng khi giải đóng gói gợi ý [33]. Việc không thực hiện phép kiểm tra này dẫn đến một lược đồ chữ ký không có tính không thể giả mạo tồn tại mạnh [34].

D.3 Những thay đổi so với dự thảo công khai đầu tiên của FIPS 204

Trong phiên bản cuối cùng của tiêu chuẩn ML-DSA, phép kiểm tra đầu vào sai định dạng bị bỏ sót đã được khôi phục trong thuật toán giải đóng gói gợi ý (**Thuật toán 21**). Ngoài ra, trong phiên bản cuối cùng của ML-DSA, tất cả các bit của $\tilde{c}$ đều được sử dụng để sinh c (**Thuật toán 29**), và ExpandMask (**Thuật toán 34**) được sửa đổi để lấy các bit đầu ra từ phần đầu của đầu ra H .

Dựa trên các ý kiến góp ý được gửi về dự thảo, Mục 7.4 bổ sung thêm chi tiết cho phiên bản băm trước HashML-DSA. Những sửa đổi này bao gồm việc phân tách miền giữa các trường hợp ký trực tiếp thông điệp và các trường hợp ký giá trị băm của thông điệp. Các thay đổi được thực hiện bằng cách định nghĩa

tường minh các hàm bên ngoài cho cả hai phiên bản của các hàm ký và xác minh; các hàm này gọi một hàm nội bộ tương ứng với các hàm ký hoặc xác minh trong dự thảo FIPS. Việc phân tách miền được đưa vào đầu vào của hàm nội bộ (xem các **Thuật toán 2, 3, 4, 5, 7 và 8**). Để đơn giản hóa các API và phục vụ mục đích kiểm thử, tiêu chuẩn này cũng đưa ra cách phân chia hàm bên ngoài/nội bộ tương tự cho việc sinh khóa (xem các **Thuật toán 1 và 6**), nhưng đây chỉ là một thay đổi về biên tập, vì thuật toán sinh khóa bên ngoài tương đương về chức năng với thuật toán sinh khóa trong dự thảo FIPS.

Cuối cùng, để tăng khả năng chống sử dụng sai trong trường hợp các khóa cho những bộ tham số khác nhau có thể được mở rộng từ cùng một mầm [35], việc phân tách miền đã được bổ sung vào dòng 1 của **Thuật toán 6**.

Thư mục tài liệu tham khảo

- [1]. National Institute of Standards and Technology (2023) Digital signature standard (DSS), (U.S. Department of Commerce, Washington, DC), Federal Information Processing Standards Publication (FIPS) 186-5. <https://doi.org/10.6028/NIST.FIPS.186-5>
- [2]. Barker E (2020) Guideline for using cryptographic standards in the federal government: Cryptographic mechanisms, (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-175B, Rev. 1 [or as amended]. <https://doi.org/10.6028/NIST.SP.800-175Br1>.
- [3]. Barker E (2006) Recommendation for obtaining assurances for digital signature applications, National Institute of Standards and Technology, Gaithersburg, MD. NIST Special Publication (SP) 800-89 [or as amended]. <https://doi.org/10.6028/NIST.SP.800-89>.
- [4]. Langlois A, Stehlé D (2015) Worst-case to average-case reductions for module lattices. Designs, Codes and Cryptography 75(3):565–599. <https://doi.org/10.1007/s10623-014-9938-4>.
- [5]. Bai S, Ducas L, Kiltz E, Lepoint T, Lyubashevsky V, Schwabe P, Seiler G, Stehlé D (2020) CRYSTALS Dilithium: Algorithm specifications and supporting documentation, Submission to the NIST's postquantum cryptography standardization process. Available at <https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions>.
- [6]. Bai S, Ducas L, Kiltz E, Lepoint T, Lyubashevsky V, Schwabe P, Seiler G, Stehlé D (2021) CRYSTALS Dilithium: Algorithm specifications and supporting documentation (Version 3.1). Available at <https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf>.
- [7]. National Institute of Standards and Technology (2015) SHA-3 standard: Permutation-based hash and extendable-output functions, (U.S. Department of Commerce, Washington, DC), Federal Information Processing Standards Publication (FIPS) 202. <https://doi.org/10.6028/NIST.FIPS.202>.
- [8]. National Institute of Standards and Technology (2015) Secure hash standard (SHS), (U.S. Department of Commerce, Washington, DC), Federal Information Processing Standards Publication (FIPS) 180-4. <https://doi.org/10.6028/NIST.FIPS.180-4>.
- [9]. Barker E (2020) Recommendation for key management: Part 1 - general, (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-57 Part 1, Rev. 5 [or as amended]. <https://doi.org/10.6028/NIST.SP.800-57pt1r5>.
- [10]. Lyubashevsky V (2009) Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. Advances in Cryptology – ASIACRYPT 2009, ed Matsui M (Springer Berlin Heidelberg, Berlin, Heidelberg), pp 598–616. https://doi.org/10.1007/978-3-642-10366-7_35.
- [11]. Lyubashevsky V (2012) Lattice signatures without trapdoors. EUROCRYPT (Springer), Lecture Notes in Computer Science, Vol. 7237, pp 738–755. https://doi.org/10.1007/978-3-642-29011-4_43
- [12]. Güneysu T, Lyubashevsky V, Pöppelmann T (2012) Practical lattice-based cryptography: A signature scheme for embedded systems. CHES (Springer), Vol. 7428, pp 530–547. https://doi.org/10.1007/978-3-642-33027-8_31.
- [13]. Bai S, Galbraith SD (2014) An improved compression technique for signatures based on learning with errors. Topics in Cryptology – CT-RSA 2014, ed Benaloh J (Springer International Publishing, Cham), pp 28–47. https://doi.org/10.1007/978-3-319-04852-9_2.

- [14]. Cremers C, Düzlül S, Fiedler R, Janson C, Fischlin M (2021) BUFFing signature schemes beyond unforgeability and the case of post-quantum signatures. 2021 IEEE Symposium on Security and Privacy (SP) (IEEE Computer Society, Los Alamitos, CA, USA), pp 1696–1714. <https://doi.org/10.1109/SP40001.2021.00093>.
- [15]. Regev O (2005) On lattices, learning with errors, random linear codes, and cryptography. Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing STOC '05 (Association for Computing Machinery, New York, NY, USA), p 84–93. <https://doi.org/10.1145/1060590.1060603>.
- [16]. Kiltz E, Lyubashevsky V, Schaffner C (2018) A concrete treatment of Fiat-Shamir signatures in the quantum random-oracle model. Advances in Cryptology – EUROCRYPT 2018, eds Nielsen JB, Rijmen V (Springer International Publishing, Cham), pp 552–586. https://doi.org/10.1007/978-3-319-78372-7_18.
- [17]. Barker E, Barker W (2019) Recommendation for key management: Part 2 - best practices for key management organizations, National Institute of Standards and Technology, Gaithersburg, MD. NIST Special Publication (SP) 800-57 Part 2, Rev. 1. <https://doi.org/10.6028/NIST.SP.800-57pt2r1>.
- [18]. Barker E, Dang Q (2019) Recommendation for key management: Part 3 - application-specific key management guidance, National Institute of Standards and Technology, Gaithersburg, MD. NIST Special Publication (SP) 800-57 Part 3, Rev. 1. <http://doi.org/10.6028/NIST.SP.800-57pt3r1>.
- [19]. Barker E, Kelsey J (2015) Recommendation for random number generation using deterministic random bit generators, (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-90A, Rev. 1. <https://doi.org/10.6028/NIST.SP.800-90Ar1>.
- [20]. Sönmez Turan M, Barker E, Kelsey J, McKay K, Baish M, Boyle M (2018) Recommendation for the entropy sources used for random bit generation, (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-90B. <https://doi.org/10.6028/NIST.SP.800-90B>.
- [21]. Barker E, Kelsey J, McKay K, Roginsky A, Turan MS (2024) Recommendation for random bit generator (RBG) constructions, (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-90C 4pd. <https://doi.org/10.6028/NIST.SP.800-90C.4pd>.
- [22]. Bruinderink LG, Pessl P (2018) Differential fault attacks on deterministic lattice signatures. IACR Transactions on Cryptographic Hardware and Embedded Systems (3):21–43. <https://doi.org/10.13154/tches.v2018.i3.21-43>.
- [23]. Poddebniak D, Somorovsky J, Schinzel S, Lochter M, Rösler P (2018) Attacking deterministic signature schemes using fault attacks. 2018 IEEE European Symposium on Security and Privacy (EuroS&P) (IEEE), pp 338–352. <https://doi.org/10.1109/EuroSP.2018.00031>.
- [24]. Samwel N, Batina L, Bertoni G, Daemen J, Susella R (2018) Breaking ed25519 in wolfssl. Topics in Cryptology–CT-RSA 2018: The Cryptographers' Track at the RSA Conference 2018, San Francisco, CA, USA, April 16-20, 2018, Proceedings (Springer), pp 1–20. https://doi.org/10.1007/978-3-319-76953-0_1.
- [25]. Kelsey J, Chang S, Perlner R (2016) SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash and ParallelHash, (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-185 [or as amended]. <https://doi.org/10.6028/NIST.SP.800-185>.
- [26]. National Institute of Standards and Technology (2016) Submission requirements and evaluation criteria for the post-quantum cryptography standardization process. Available at

- <https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf>.
- [27]. Alagic G, Apon D, Cooper D, Dang Q, Dang T, Kelsey J, Lichtinger J, Liu YK, Miller C, Moody D, Peralta R, Perlner R, Robinson A, Smith-Tone D (2022) Status report on the third round of the NIST post-quantum cryptography standardization process (National Institute of Standards and Technology, Gaithersburg, MD), NIST Interagency or Internal Report (IR) 8413. <https://doi.org/10.6028/NIST.IR.8413-upd1>.
- [28]. Avanzi R, Bos J, Ducas L, Kiltz E, Lepoint T, Lyubashevsky V, Schanck JM, Schwabe P, Seiler G, Stehlé D (2020) CRYSTALS-Kyber algorithm specifications and supporting documentation, 3rd Round submission to the NIST's post-quantum cryptography standardization process. Available at <https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions>.
- [29]. Housley R (2009) Cryptographic Message Syntax (CMS), Internet Engineering Task Force (IETF) request for comments (RFC) 5652, <https://doi.org/10.17487/RFC5652>.
- [30]. Schnorr C (1990) Efficient identification and signatures for smart cards. Advances in Cryptology — CRYPTO' 89 Proceedings, ed Brassard G (Springer New York, New York, NY), pp 239–252. https://doi.org/10.1007/0-387-34805-0_22.
- [31]. Josefsson S, Liusvaara I (2017) Edwards-Curve Digital Signature Algorithm (EdDSA), RFC 8032. <https://doi.org/10.17487/RFC8032>.
- [32]. Lyubashevsky V (2021) Round 3 Official Comment: CRYSTALS-DILITHIUM. Available at <https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/BjfrMIIdnhM/m/W7kkVOFDBAAJ>.
- [33]. Hamburg M (2024) Dilithium hint unpacking. Available at <https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/TQo-qFbBO1A/m/YcYKjMbIAAAJ>.
- [34]. Mattsson (on behalf of Sönke Jendral) JP (2024) Dilithium hint unpacking. Available at <https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/TQo-qFbBO1A/m/sLjseYISAwAJ>.
- [35]. Lee S (2024) Updates for FIPS 203. Available at <https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/Rb0nFvffTEQ/m/lw-k7tVdBQAJ>.
- [36]. TCVN 12853:2020: Các kỹ thuật an toàn – Bộ tạo bit ngẫu nhiên.
- [37]. TCVN 11816-3:2017: Công nghệ thông tin – Các kỹ thuật an toàn – Hàm băm – Phần 3: Hàm băm chuyên dụng.