

TCVN

TIÊU CHUẨN QUỐC GIA

DỰ THẢO

TCVN XXXXX:2026

**CÔNG NGHỆ THÔNG TIN – CÁC KỸ THUẬT AN TOÀN –
CÁC THUẬT TOÁN MÃ HÓA –
PHẦN 2: MẬT MÃ PHI ĐỐI XỨNG
BỔ SUNG 2**

Information technology – Security techniques – Encryption algorithms –

Part 2: Asymmetric cipher

Amendment 2

HÀ NỘI - 2026

Mục lục

1. Phạm vi áp dụng	5
2. Tài liệu viện dẫn	5
3. Các yêu cầu	5
4. Thuật ngữ, ký hiệu và định nghĩa	5
4.1. Thuật ngữ	5
4.2. Ký hiệu	7
4.3. Diễn giải đặc tả thuật toán	9
5. Số học và các biểu diễn	9
5.1. Các tham số	9
5.2. Số học với mảng số nguyên	9
5.3. Số học với đa thức và biểu diễn NTT	9
5.4. Số học với ma trận và vec-tơ	10
5.5. Áp dụng các thuật toán cho mảng	12
6. Các thuật toán hỗ trợ	12
6.1. Chuyển đổi giữa bit và byte	12
6.1.1. BitsToBytes	12
6.1.2. BytesToBits	12
6.2. Các hàm mật mã	13
6.3. Các thuật toán nén và giải nén	14
6.4. Mã hóa biểu diễn và giải mã biểu diễn	14
6.4.1. Các tham số	14
6.4.2. Mã hóa biểu diễn	15
6.4.3. Giải mã biểu diễn	15
6.5. Các thuật toán lấy mẫu	16
6.5.1. Tổng quan	16
6.5.2. $\text{SampleNTT}(B)$	16
6.5.3. $\text{SamplePolyCBD}\eta(B)$	16
6.6. Phép biến đổi lý thuyết số	17
6.6.1. Tổng quan	17
6.6.2. $\text{NTT}(f)$	18
6.6.3. $\text{NTT}^{-1}(f)$	18
6.7. Phép nhân trong miền NTT	19
6.7.1. Tổng quan	19

6.7.2. $\text{MultiplyNTTs}(f, g)$	19
6.7.3. $\text{BaseCaseMultiply}(a_0, a_1, b_0, b_1, \gamma)$	20
7. Lược đồ thành phần K-PKE	20
7.1. Tổng quan	20
7.2. Sinh khóa K-PKE	21
7.2.1. Tổng quan	21
7.2.2. $\text{K-PKE.KeyGen}(d)$	21
7.3 Mã hóa K-PKE	22
7.3.1. Tổng quan	22
7.3.2. $\text{K-PKE.Encrypt}(ek_{PKE}, m, r)$	23
7.4 Giải mã K-PKE	24
7.4.1 Tổng quan	24
7.4.2 $\text{K-PKE.Decrypt}(dk_{PKE}, c)$	24
8. Cơ chế đóng gói khóa	24
8.1. Tổng quan	24
8.2. Sinh khóa	24
8.2.1. Tổng quan	24
8.2.2. $\text{ML-KEM.KeyGen}()$	25
8.3. Đóng gói khóa	25
8.4. Giải gói khóa	25
9. Các bộ tham số	26
Phụ lục A	27
A.1. Các mức an toàn và tài liệu hỗ trợ	27
A.2. Lỗi giải gói khóa	27
Tài liệu tham khảo	29

Lời nói đầu

Tiêu chuẩn TCVN XXXX:2026 CƠ CHẾ ĐÓNG GÓI KHÓA DỰA TRÊN LƯỚI MÔ-ĐUN ML-KEM 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 XXXX:2026 CƠ CHẾ ĐÓNG GÓI KHÓA DỰA TRÊN LƯỚI MÔ-ĐUN ML-KEM được xây dựng trên cơ sở chấp nhận hoàn toàn tương đương với tiêu chuẩn ISO/IEC 18033-2:2006/Amd. 2:2026.

1. Phạm vi áp dụng

Tiêu chuẩn này đặc tả cơ chế đóng gói khóa dựa trên lưới mô-đun ML-KEM được xây dựng dựa trên bài toán học với lỗi dạng mô-đun (MLWE). Cơ chế này được thiết kế nhằm cung cấp khả năng bảo vệ trước các tấn công có năng lực lượng tử. Tiêu chuẩn quy định hai bộ tham số cho ML-KEM, cung cấp các sự đánh đổi khác nhau giữa độ an toàn và hiệu năng. ML-KEM có thể được sử dụng như một thành phần trong các giao thức và hệ thống mật mã nhằm thiết lập khóa bí mật dùng chung cho việc bảo vệ thông tin.

Lưu ý: ML-KEM được kế thừa và phát triển từ cơ chế KEM CRYSTALS-KYBER.

2. Tài liệu viện dẫn

Tiêu chuẩn này tham chiếu đến các tiêu chuẩn về thuật toán và tham số mật mã sau.

- ISO/IEC 10118-3:2018 Công nghệ thông tin – Các kỹ thuật an toàn – Hàm băm – Phần 3: Hàm băm chuyên dụng.
- ISO/IEC 18031:2025 Công nghệ thông tin — Kỹ thuật an toàn — Tạo bit ngẫu nhiên.

3. Các yêu cầu

Để được coi phù hợp với tài liệu này đối với ML-KEM, một thuật toán phải đáp ứng các yêu cầu sau:

- 1) chỉ rõ tên của một trong các hàm: ML-KEM.KeyGen, ML-KEM.Encaps hoặc ML-KEM.Decaps;
- 2) xác định chính xác một bộ tham số được liệt kê trong mục 9; và
- 3) tính toán chính xác hàm toán học tương ứng được định nghĩa trong tài liệu này cho bộ tham số đó.

Ví dụ: Một triển khai hàm ML-KEM.KeyGen dành cho bộ tham số ML-KEM-768 phải tính toán chính xác hàm ML-KEM.KeyGen quy định riêng cho bộ tham số đó. Cụ thể, triển khai đó phải đọc chính xác 256 bit ngẫu nhiên và phải tạo ra đầu ra hoàn toàn trùng khớp với kết quả mà thuật toán ML-KEM.KeyGen quy định bên dưới tạo ra khi được cung cấp cùng một chuỗi 256 bit đó.

Lược đồ mã hóa khóa công khai K-PKE được mô tả trong tiêu chuẩn này không được phép sử dụng như một lược đồ mật mã độc lập. Thay vào đó, các thuật toán cấu thành nên K-PKE chỉ được sử dụng làm chương trình con trong các thuật toán lớn hơn của ML-KEM.

Các triển khai tuân thủ tiêu chuẩn phải sử dụng các quy tắc biểu diễn dữ liệu được quy định chi tiết trong tiêu chuẩn này.

4. Thuật ngữ, ký hiệu và định nghĩa

4.1. Thuật ngữ

Với mục đích của tiêu chuẩn này, các thuật ngữ và định nghĩa sau được áp dụng.

Bản mã KEM

Một chuỗi bit được tạo ra bởi quá trình đóng gói khóa và được sử dụng làm đầu vào cho quá trình giải gói khóa.

Giải gói khóa

Quá trình áp dụng thuật toán Decaps của KEM. Thuật toán này nhận đầu vào là bản mã KEM và khóa giải gói, và đưa ra đầu ra là khóa bí mật được chia sẻ.

Khóa giải gói

Khóa mật mã được tạo ra bởi KEM trong quá trình sinh khóa và được sử dụng trong quá trình giải gói khóa. Khóa giải gói phải được giữ bí mật và phải bị hủy bỏ sau khi không còn cần thiết nữa.

Khóa giải mã

Khóa mật mã được sử dụng với một PKE để giải mã bản mã thành bản rõ. Khóa giải mã phải được giữ bí mật và phải bị hủy bỏ sau khi không còn cần thiết nữa.

Hủy bỏ

Một thao tác được áp dụng cho khóa hoặc một phần dữ liệu bí mật khác. Sau khi một phần dữ liệu bí mật bị phá hủy, không thể khôi phục được bất kỳ thông tin nào về giá trị của nó.

Đóng gói khóa

Quá trình áp dụng thuật toán Encaps của KEM. Thuật toán này nhận khóa đóng gói làm đầu vào và tạo ra một khóa bí mật được chia sẻ cùng với bản mã tương ứng làm đầu ra.

Khóa đóng gói

Khóa mật mã được sinh ra bởi KEM trong quá trình sinh khóa và được sử dụng trong quá trình đóng gói khóa. Khóa đóng gói có thể được công khai.

Khóa mã hóa

Khóa mật mã được sử dụng với một PKE để mã hóa bản rõ thành bản mã. Khóa mã hóa có thể được công khai.

Hàm băm

Một hàm trên chuỗi bit với độ dài đầu ra được cố định. Các hàm băm được chấp thuận trong tiêu chuẩn này phải đáp ứng các thuộc tính sau:

1. (Một chiều) Về mặt tính toán, không thể tìm thấy bất kỳ đầu vào nào tương ứng với bất kỳ đầu ra mới nào được xác định trước.
2. (Kháng va chạm) Về mặt tính toán, không thể tìm thấy hai đầu vào khác nhau nào có cùng đầu ra.

Khóa

Chuỗi bit được sử dụng kết hợp với một thuật toán mật mã, chẳng hạn như khóa đóng gói và khóa giải gói (của KEM), khóa bí mật được chia sẻ (được tạo ra bởi KEM), khóa mã hóa và khóa giải mã (của PKE).

Cơ chế đóng gói khóa (KEM)

Một tập hợp gồm ba thuật toán mật mã (KeyGen, Encaps, Decaps) có thể được hai bên sử dụng để thiết lập một khóa bí mật được chia sẻ trên kênh công khai.

Thiết lập khóa

Một quy trình tạo ra nguyên liệu khóa bí mật được chia sẻ giữa các bên khác nhau.

Cặp khóa

Một bộ gồm hai khóa với thuộc tính là một khóa có thể được công khai trong khi khóa còn lại phải được giữ bí mật. Trong tiêu chuẩn này, cặp khóa có thể đề cập đến khóa đóng gói và khóa giải gói của KEM hoặc khóa mã hóa và khóa giải mã của PKE.

Bên

Một cá nhân, tổ chức, thiết bị hoặc quy trình. Trong đặc tả này, có hai bên (ví dụ: Bên A và Bên B, hoặc Alice và Bob) cùng thực hiện quy trình thiết lập khóa bằng cách sử dụng KEM.

Giả ngẫu nhiên

Một quy trình (hoặc dữ liệu do quy trình đó tạo ra) được gọi là giả ngẫu nhiên khi kết quả là bất định nhưng đồng thời cũng có vẻ ngẫu nhiên miễn là hoạt động bên trong của quy trình không bị quan sát thấy. Đối với mục đích mật mã, "ngẫu nhiên hiệu quả" có nghĩa là "không thể phân biệt được về mặt tính toán với ngẫu nhiên trong giới hạn của mức độ an toàn dự định".

Kênh công khai

Kênh liên lạc giữa hai bên. Kênh liên lạc này có thể bị theo dõi và thậm chí bị can thiệp bởi bên thứ ba.

Lược đồ mã hóa khóa công khai (PKE)

Một tập gồm ba thuật toán mật mã (KeyGen, Encrypt, Decrypt) có thể được hai bên sử dụng để gửi dữ liệu bí mật qua kênh công khai. Còn được gọi là lược đồ mã hóa phi đối xứng.

Khóa bí mật được chia sẻ

Một giá trị bí mật được tính toán trong quá trình thiết lập khóa, được cả hai bên tham gia biết, có thể được sử dụng trực tiếp làm khóa mật mã trong mật mã đối xứng. Khóa bí mật được chia sẻ phải được giữ kín và phải bị hủy bỏ khi không còn cần thiết nữa.

Mức an toàn

Một con số liên quan đến độ mạnh an toàn của thuật toán mật mã hậu lượng tử, theo quy định của NIST.

Độ mạnh an toàn

Một con số liên quan đến khối lượng công việc (tức là số lượng thao tác) cần thiết để phá vỡ một thuật toán hoặc hệ thống mật mã.

4.2. Ký hiệu

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

Ký hiệu	Định nghĩa
$A \parallel B$	Phép nối hai mảng hoặc hai chuỗi bit A và B .
$\mathbb{B}$	Tập hợp $\{0,1, \dots, 255\}$ gồm các số nguyên không dấu 8-bit (các byte).

$B[i]$	Phần tử tại chỉ số i trong mảng B . Tất cả các mảng đều có chỉ số bắt đầu từ số 0.
$B[k:m]$	Mảng con $(B[k], B[k+1], \dots, B[m-1])$ của mảng B .
$\text{BitRev}_7(r)$	Phép đảo ngược bit của một số nguyên 7-bit r . Cụ thể, nếu $r = r_0 + 2r_1 + 4r_2 + \dots + 64r_6$ với $r_i \in \{0,1\}$, thì $\text{BitRev}_7(r) = r_6 + 2r_5 + 4r_4 + \dots + 64r_0$.
n	Số nguyên 256 xuyên suốt tiêu chuẩn này.
q	Số nguyên tố $3329 = 2^8 \cdot 13 + 1$ xuyên suốt tiêu chuẩn này.
$\mathbb{Q}$	Tập hợp các số hữu tỉ.
$r \bmod m$	Số nguyên duy nhất r' trong tập $\{0,1, \dots, m-1\}$ sao cho $r - r'$ chia hết m .
$s \leftarrow x$	Biến s được gán giá trị của biểu thức x .
$s \overset{\$}{\leftarrow} \mathbb{B}^l$	Biến s được gán giá trị là một mảng gồm l byte ngẫu nhiên. Các byte này phải được khởi tạo bằng nguồn ngẫu nhiên.
S^*	Tập hợp các bộ (hoặc mảng) có độ dài hữu hạn gồm các phần tử thuộc tập hợp S , bao gồm cả bộ rỗng (hoặc mảng rỗng).
S^k	Tập hợp các bộ k phần tử (hoặc mảng có độ dài k) gồm các phần tử thuộc tập hợp S .
$\mathbf{v}^T, \mathbf{A}^T$	Chuyển vị của một vector hàng hoặc vector cột $\mathbf{v}$; đồng thời là ma trận chuyển vị của một ma trận $\mathbf{A}$.
$\mathbb{Z}$	Tập hợp các số nguyên.
$\lceil x \rceil$	Phép làm tròn x đến số nguyên gần nhất; nếu $x = y + \frac{1}{2}$ với một $y \in \mathbb{Z}$ nào đó, thì $\lceil x \rceil = y + 1$.
$\lfloor x \rfloor$	Biểu thị số nguyên lớn nhất không vượt quá x .
$\mathcal{D}_\eta(R_q)$	Một phân bố xác suất của đa thức trên R_q với các hệ số nhỏ, được sử dụng để lấy mẫu nhiều. Phân bố được tham số bởi η .
$\mathbb{Z}_m$	Vành các số nguyên theo mô-đun m , tức là tập hợp $\{0,1, \dots, m-1\}$ đi kèm với các phép toán cộng và nhân theo mô-đun m .
R_q	Vành $\mathbb{Z}_q[X]/(X^n + 1)$ bao gồm các đa thức có dạng $f = f_0 + f_1X + \dots + f_{255}X^{255}$ trong đó $f_j \in \mathbb{Z}_q$ với mọi j , đi kèm với phép cộng và phép nhân theo mô-đun $X^n + 1$.
f_j	Hệ số của X^j trong đa thức $f = f_0 + f_1X + \dots + f_{255}X^{255} \in R_q$.
T_q	Ảnh của vành R_q qua phép biến đổi lý thuyết số (NTT). Các phần tử của nó được gọi là "biểu diễn NTT" của các đa thức trong R_q .
$\hat{f}$	Phần tử thuộc T_q tương đương với biểu diễn NTT của một đa thức $f \in R_q$.

4.3. Diễn giải đặc tả thuật toán

Đối với hầu hết các biến trong thuật toán, kiểu dữ liệu (ví dụ: bit, byte, mảng các bit) có thể dễ dàng suy ra từ ngữ cảnh. Đối với các biến đại diện cho đầu vào hoặc đầu ra của một thuật toán, kiểu dữ liệu sẽ được mô tả rõ ràng ở phần bắt đầu của thuật toán đó. Đối với tất cả các biến khác, kiểu dữ liệu đều rõ ràng theo ngữ cảnh. Trong một thuật toán đơn lẻ, kiểu dữ liệu của một biến được xác định vào lần đầu tiên biến đó được sử dụng và nó không thay đổi. Tên biến có thể được tái sử dụng trên các thuật toán khác nhau, bao gồm cả việc mang các kiểu dữ liệu khác nhau. Bên cạnh các kiểu dữ liệu tiêu chuẩn (ví dụ: bit, byte) và cấu trúc dữ liệu (ví dụ: mảng), các số nguyên đồng dư theo mô-đun m (tức là các phần tử của $\mathbb{Z}_m$) cũng được sử dụng như một kiểu dữ liệu trừu tượng. Phép toán rút gọn theo mô-đun m được thực hiện ngầm định bất cứ khi nào có một phép gán cho một biến thuộc $\mathbb{Z}_m$. Ví dụ, đối với $z \in \mathbb{Z}_m$ và bất kỳ số nguyên x, y nào, câu lệnh $z \leftarrow x + y$ có nghĩa là z được gán giá trị của $(x + y) \bmod m$. Đặc tả thuật toán không ràng buộc cách một số nguyên theo mô-đun m được biểu diễn trong các triển khai thực tế hoặc cách tính toán phép rút gọn mô-đun, trừ khi được quy định rõ ràng.

5. Số học và các biểu diễn

5.1. Các tham số

ML-KEM được quy định trong tiêu chuẩn này với hai bộ tham số được ký hiệu như sau:

- ML-KEM-768;

- ML-KEM-1024.

Các bộ tham số này được mô tả và thảo luận chi tiết trong mục 9. Mỗi bộ tham số sẽ gán các giá trị số cụ thể cho năm biến số nguyên: k , η_1 , η_2 , d_u và d_v . Giá trị của các biến này trong từng bộ tham số được cung cấp trong Bảng 1. Ngoài năm tham số thay đổi này, còn có hai hằng số cố định: $n = 256$ và $q = 3329$.

5.2. Số học với mảng số nguyên

Cho $a \in \mathbb{Z}_m$ và $X, Y \in \mathbb{Z}_m^l$. Các câu lệnh:

$$Z \leftarrow a \cdot X$$

$$W \leftarrow X + Y$$

sẽ tạo ra hai mảng $Z, W \in \mathbb{Z}_m^l$, với tính chất là $Z[i] = a \cdot X[i]$ và $W[i] = X[i] + Y[i]$ với mọi i .

Phép nhân các mảng trong $\mathbb{Z}_m^l$ chỉ có ý nghĩa khi $m = q$ và $l = n = 256$, trong trường hợp đó nó tương ứng với phép nhân trong vành R_q . Phép toán này được mô tả chi tiết trong mục 5.3.

5.3. Số học với đa thức và biểu diễn NTT

Một phép toán thiết yếu trong ML-KEM là phép biến đổi lý thuyết số (Number-Theoretic Transform - NTT), ánh xạ một đa thức f trong một vành R_q nhất định sang “biểu diễn NTT” của nó, ký hiệu là $\hat{f}$, trong một vành khác là T_q . Tài liệu này biểu diễn các phần tử của R_q và T_q bằng cách sử dụng các mảng số nguyên theo mô-đun q như sau.

Một phần tử f của R_q là một đa thức có dạng:

$$f = f_0 + f_1X + \dots + f_{255}X^{255} \in R_q$$

và được biểu diễn trong đặc tả thuật toán bằng một mảng

$$(f_0, f_1, \dots, f_{255}) \in \mathbb{Z}_q^{256}$$

chứa các hệ số của f . Để đơn giản hóa ký hiệu, mảng này cũng được ký hiệu là f . Do đó, phần tử thứ i của mảng f sẽ chứa hệ số thứ i của đa thức f (tức là $f[i] = f_i$).

Một phần tử $\hat{g}$ của T_q (đôi khi được gọi là “biểu diễn NTT”) là một bộ gồm 128 đa thức, mỗi đa thức có bậc tối đa là một. Cụ thể:

$$\hat{g} = (\hat{g}_{0,0} + \hat{g}_{0,1}X, \hat{g}_{1,0} + \hat{g}_{1,1}X, \dots, \hat{g}_{127,0} + \hat{g}_{127,1}X) \in T_q.$$

Một đối tượng đại số như vậy được biểu diễn trong đặc tả thuật toán bằng một mảng

$$(\hat{g}_{0,0}, \hat{g}_{0,1}, \hat{g}_{1,0}, \hat{g}_{1,1}, \dots, \hat{g}_{127,0}, \hat{g}_{127,1}) \in \mathbb{Z}_q^{256}.$$

Để đơn giản hóa ký hiệu, mảng này cũng được ký hiệu là $\hat{g}$. Trong trường hợp này, phép ánh xạ giữa các phần tử của mảng và các hệ số được quy định là $\hat{g}[2i] = \hat{g}_{i,0}$ và $\hat{g}[2i+1] = \hat{g}_{i,1}$ với mọi $i \in \{0, 1, \dots, 127\}$.

Việc chuyển đổi qua lại giữa một đa thức $f \in R_q$ và biểu diễn NTT $\hat{f} \in T_q$ của nó được thực hiện thông qua các thuật toán NTT và NTT^{-1} . Các thuật toán này tác động trực tiếp trên các mảng hệ số như đã mô tả ở trên và thỏa mãn $\hat{f} = \text{NTT}(f)$ và $f = \text{NTT}^{-1}(\hat{f})$.

Các phép toán đại số gồm phép cộng và phép nhân vô hướng trong R_q và T_q được thực hiện theo từng tọa độ. Ví dụ, nếu $a \in \mathbb{Z}_q$ và $f \in R_q$, thì hệ số thứ i của đa thức $a \cdot f \in R_q$ bằng $a \cdot f_i \bmod q$. Trong đặc tả thuật toán, các phần tử của R_q và T_q đều được biểu diễn bằng các mảng hệ số (tức là các phần tử thuộc $\mathbb{Z}_q^{256}$) như đã nêu trên. Do đó, các phép toán đại số gồm phép cộng và phép nhân vô hướng được thực hiện bằng cách cộng và nhân vô hướng các mảng hệ số tương ứng. Ví dụ, phép cộng hai biểu diễn NTT trong đặc tả thuật toán được thực hiện bằng một câu lệnh có dạng $\hat{h} \leftarrow \hat{f} + \hat{g}$, trong đó $\hat{h}, \hat{f}, \hat{g} \in \mathbb{Z}_q^{256}$ là các mảng hệ số.

Các phép toán đại số liên quan đến phép nhân trong R_q và T_q được xử lý như sau. Vì lý do hiệu năng, phép nhân trong R_q không được sử dụng. Ý nghĩa đại số của phép nhân trong T_q được thảo luận chi tiết trong mục 6.6. Trong đặc tả thuật toán, phép toán này được thực hiện bởi thuật toán `MultiplyNTTs`. Cụ thể, nếu $\hat{f}, \hat{g} \in \mathbb{Z}_q^{256}$ là một cặp mảng (mỗi mảng biểu diễn kết quả NTT của một đa thức nào đó), thì:

$$\hat{h} \leftarrow \hat{f} \times_{T_q} \hat{g} \text{ có nghĩa là } \hat{h} \leftarrow \text{MultiplyNTTs}(\hat{f}, \hat{g}).$$

Kết quả thu được là một mảng $\hat{h} \in \mathbb{Z}_q^{256}$.

5.4. Số học với ma trận và vec-tơ

Bên cạnh các mảng số nguyên theo mô-đun q , các đặc tả thuật toán cũng sử dụng các mảng mà bản thân mỗi phần tử trong đó lại là một phần tử của $\mathbb{Z}_q^{256}$. Ví dụ, một phần tử $v \in (\mathbb{Z}_q^{256})^3$ là một mảng có độ dài bằng 3, trong đó các phần tử $v[0]$, $v[1]$ và $v[2]$ chính là các mảng thuộc $\mathbb{Z}_q^{256}$. Chúng ta có thể coi mỗi phần tử này đại diện cho một đa thức trong R_q , từ đó bản thân v đại diện cho một phần tử của mô-

đun R_q^3 .

Khi các mảng được sử dụng để đại diện cho ma trận và vector có các phần tử thuộc R_q , chúng sẽ được ký hiệu bằng các chữ cái in đậm (ví dụ: $\mathbf{v}$ cho vector và $\mathbf{A}$ cho ma trận). Khi các mảng được sử dụng để đại diện cho ma trận và vector có các phần tử thuộc T_q , chúng sẽ được ký hiệu kèm theo một dấu “mũ” (ví dụ: $\hat{\mathbf{v}}$ và $\hat{\mathbf{A}}$). Trừ khi phép toán chuyển vị được thực hiện một cách tường minh, các vector luôn được ngầm hiểu là các vector cột. Do đó, có thể xem vector là một trường hợp đặc biệt của ma trận khi chỉ có duy nhất một cột.

Việc chuyển đổi qua lại giữa các ma trận trên vành R_q và các ma trận trên vành T_q được thực hiện theo từng tọa độ. Cụ thể, nếu $\mathbf{A} \in (\mathbb{Z}_q^{256})^{k \times l}$, thì câu lệnh:

$$\hat{\mathbf{A}} \leftarrow NTT(\mathbf{A})$$

sẽ tạo ra kết quả $\hat{\mathbf{A}} \in (\mathbb{Z}_q^{256})^{k \times l}$ sao cho $\hat{\mathbf{A}}[i, j] = NTT(\mathbf{A}[i, j])$ với mọi chỉ số i, j . Quá trình này đòi hỏi phải chạy thuật toán NTT tổng cộng $k \cdot l$ lần. Trường hợp đối với vector sẽ tương ứng với $l = 1$.

Dưới đây mô tả cách thức thực hiện các phép toán số học với ma trận, trong đó vector tiếp tục được xem là một trường hợp đặc biệt của ma trận.

Phép cộng và phép nhân vô hướng được thực hiện theo từng tọa độ. Phép cộng các ma trận trên vành R_q và T_q được thực hiện một cách trực tiếp. Trong trường hợp của vành T_q , phép nhân vô hướng được thực hiện bằng cách sử dụng phép nhân trong T_q đã được mô tả ở mục 5.3. Ví dụ, nếu $\hat{f} \in \mathbb{Z}_q^{256}$ và $\hat{\mathbf{u}}, \hat{\mathbf{v}} \in (\mathbb{Z}_q^{256})^k$, thì các câu lệnh:

$$\hat{\mathbf{w}} \leftarrow \hat{f} \cdot \hat{\mathbf{u}}$$

$$\hat{\mathbf{z}} \leftarrow \hat{\mathbf{u}} + \hat{\mathbf{v}}$$

sẽ tạo ra kết quả $\hat{\mathbf{w}}, \hat{\mathbf{z}} \in (\mathbb{Z}_q^{256})^k$ thỏa mãn $\hat{\mathbf{w}}[i] = \hat{f} \times_{T_q} \hat{\mathbf{u}}[i]$ và $\hat{\mathbf{z}}[i] = \hat{\mathbf{u}}[i] + \hat{\mathbf{v}}[i]$ với mọi chỉ số i . Lưu ý rằng phép nhân và phép cộng của từng phần tử riêng lẻ ở đây được thực hiện bằng các phép toán số học phù hợp dành cho mảng hệ số của các phần tử thuộc T_q .

Việc nhân các ma trận có các phần tử trong T_q cũng là một phép toán cần thiết. Phép toán này được thực hiện bằng cách sử dụng phép nhân ma trận tiêu chuẩn, với phép nhân cơ sở (tức là phép nhân giữa các phần tử riêng lẻ) là phép nhân trong vành T_q . Nếu $\hat{\mathbf{A}}$ và $\hat{\mathbf{B}}$ là hai ma trận có các phần tử thuộc T_q , tích ma trận của chúng được ký hiệu là $\hat{\mathbf{A}} \circ \hat{\mathbf{B}}$. Dưới đây là một số ví dụ liên quan đến phép nhân ma trận. Trong các ví dụ này, $\hat{\mathbf{A}}$ là một ma trận kích thước $k \times k$, trong khi $\hat{\mathbf{u}}$ và $\hat{\mathbf{v}}$ là các vector có độ dài k . Cả ba đối tượng này đều được biểu diễn trong đặc tả thuật toán bằng các mảng: một mảng kích thước $k \times k$ cho $\hat{\mathbf{A}}$ và các mảng độ dài k cho $\hat{\mathbf{u}}$ và $\hat{\mathbf{v}}$. Các phần tử của $\hat{\mathbf{A}}, \hat{\mathbf{u}}$, và $\hat{\mathbf{v}}$ là các phần tử thuộc $\mathbb{Z}_q^{256}$. Hai câu lệnh thuật toán đầu tiên tạo ra một vector mới có độ dài k với các phần tử được quy định ở vế phải. Câu lệnh thuật toán thứ ba tính toán một tích trong; do đó kết quả sẽ nằm trong vành cơ sở (tức là T_q), và được đại diện bởi một phần tử $\hat{z}$ thuộc $\mathbb{Z}_q^{256}$:

$$\hat{\mathbf{w}} \leftarrow \hat{\mathbf{A}} \circ \hat{\mathbf{u}} \quad \hat{w}[i] = \sum_{j=0}^{k-1} \hat{\mathbf{A}}[i, j] \times_{T_q} \hat{\mathbf{u}}[j]$$

$$\hat{\mathbf{y}} \leftarrow \hat{\mathbf{A}}^T \circ \hat{\mathbf{u}} \quad \hat{\mathbf{y}}[i] = \sum_{j=0}^{k-1} \hat{\mathbf{A}}[j, i] \times_{T_q} \hat{\mathbf{u}}[j]$$

$$\hat{\mathbf{z}} \leftarrow \hat{\mathbf{u}}^T \circ \hat{\mathbf{v}} \quad \hat{\mathbf{z}} = \sum_{j=0}^{k-1} \hat{\mathbf{u}}[j] \times_{T_q} \hat{\mathbf{v}}[j]$$

Phép nhân $\times_{T_q}$ của từng phần tử riêng lẻ ở trên được thực hiện thông qua hàm MultiplyNTTs, như đã mô tả trong mục 5.3.

5.5. Áp dụng các thuật toán cho mảng

Các quy ước về số học theo từng tọa độ mô tả ở trên cũng được mở rộng cho các thuật toán tác động trên (và/hoặc tạo ra) một kiểu dữ liệu tiêu chuẩn. Khi thuật toán được gọi với đầu vào là một mảng, điều đó có nghĩa là thuật toán sẽ được gọi lặp đi lặp lại cho từng phần tử của mảng đó. Ví dụ, hàm $\text{Compress}_d: \mathbb{Z}_q \rightarrow \mathbb{Z}_{2^d}$ được định nghĩa trong mục 6.3 có thể được gọi với đầu vào là một mảng $F \in \mathbb{Z}_q^{256}$ thông qua câu lệnh:

$$K \leftarrow \text{Compress}_d(F).$$

Kết quả là $K \in \mathbb{Z}_{2^d}^{256}$ và $K[i] = \text{Compress}_d(F[i])$ với mọi chỉ số i . Phép tính toán này bao gồm việc chạy thuật toán Compress tổng cộng 256 lần.

6. Các thuật toán hỗ trợ

6.1. Chuyển đổi giữa bit và byte

6.1.1. BitsToBytes

Thuật toán BitsToBytes chuyển đổi một chuỗi bit (có độ dài là bội số của tám) thành một mảng các byte. Thuật toán nhận đầu vào là một mảng bit $b \in \{0,1\}^{8 \cdot \ell}$ và đưa ra một mảng byte $B \in \mathbb{B}^\ell$ như sau:

1. $B \leftarrow (0, \dots, 0)$
2. For $i = 0$ to $8\ell - 1$ do:
3. $B \left[\left\lfloor \frac{i}{8} \right\rfloor \right] \leftarrow B \left[\left\lfloor \frac{i}{8} \right\rfloor \right] + b[i] \cdot 2^{i \bmod 8}$
4. End for
5. Return B

6.1.2. BytesToBits

Thuật toán BytesToBits thực hiện quá trình ngược lại của BitsToBytes, chuyển đổi một mảng byte thành một mảng bit. Thuật toán nhận đầu vào là một mảng byte $B \in \mathbb{B}^\ell$ và đưa ra một mảng bit $b \in \{0,1\}^{8 \cdot \ell}$ như sau:

1. $C \leftarrow B$
2. For $i = 0$ to $\ell - 1$ do:
3. For $j = 0$ to 7 do:

4. $b[8i + j] \leftarrow C[i] \bmod 2$
5. $C[i] \leftarrow \left\lfloor \frac{C[i]}{2} \right\rfloor$
6. End for
7. End for
8. Return b

6.2. Các hàm mật mã

Các thuật toán được quy định trong tiêu chuẩn này yêu cầu sử dụng một số hàm mật mã. Mỗi hàm phải được triển khai bằng một hàm băm hoặc một hàm đầu ra mở rộng (XOF), như được quy định dưới đây. Các hàm băm và XOF liên quan được quy định chi tiết trong tiêu chuẩn ISO/IEC 10118-3:2018 và được sử dụng như sau.

SHA3-256 và SHA3-512 là các hàm băm với đầu vào có độ dài biến đổi và đầu ra có độ dài cố định. Trong tiêu chuẩn này, các lệnh gọi hàm đối với đầu vào M lần lượt được ký hiệu là $\text{SHA3-256}(M)$ và $\text{SHA3-512}(M)$.

SHAKE128 và SHAKE256 là các hàm XOF với đầu vào có độ dài biến đổi và đầu ra có độ dài biến đổi. Các lệnh gọi hàm này đối với đầu vào M được ký hiệu theo hai cách khác nhau, tùy thuộc vào việc độ dài đầu ra mong muốn ℓ (tính bằng bit) đã được biết tại thời điểm gọi hàm hay chưa. Trong tiêu chuẩn này, quy ước ký hiệu như sau: Nếu ℓ đã biết tại thời điểm gọi hàm, lệnh gọi được ký hiệu là $\text{SHAKE128}(M, \ell)$ hoặc $\text{SHAKE256}(M, \ell)$. Đối với SHAKE128, đôi khi độ dài đầu ra không được biết tại thời điểm gọi; trong những trường hợp đó, lệnh gọi được ký hiệu là $\text{SHAKE128}(M)$ và tiến trình băm sẽ hoạt động giống như một luồng dữ liệu byte cung cấp các byte giả ngẫu nhiên (bằng cách thực hiện thêm các vòng nén “squeezing” như quy định trong ISO/IEC 10118-3:2018, C.2) cho đến khi không cần thêm byte nào nữa.

Các hàm trên đóng nhiều vai trò khác nhau trong các thuật toán được quy định ở tài liệu này. Mỗi hàm được gán một ký hiệu cụ thể như sau:

- Hàm giả ngẫu nhiên (PRF): Hàm PRF nhận một tham số η (có giá trị bằng 2 đối với các bộ tham số được xem xét trong tài liệu này), một đầu vào 32-byte, và một đầu vào 1-byte, được diễn giải dưới dạng các chuỗi bit. Nó tạo ra một đầu ra có độ dài $64 \cdot \eta$ byte. Hàm được ký hiệu là $\text{PRF}_\eta: \mathbb{B}^{32} \times \mathbb{B} \rightarrow \mathbb{B}^{64\eta}$, và phải được triển khai như sau:

$$\text{PRF}_\eta(s, b) = \text{SHAKE256}(s \parallel b, 8 \cdot 64 \cdot \eta),$$

trong đó $\eta = 2$, $s \in \mathbb{B}^{32}$, và $b \in \mathbb{B}$. Tại đây, η chỉ được sử dụng để chỉ định độ dài đầu ra mong muốn chứ không thực hiện phân tách miền. Lưu ý rằng tham số độ dài đầu ra cho SHAKE256 được chỉ định bằng đơn vị bit. Đầu vào $s \in \mathbb{B}^{32}$ và $b \in \mathbb{B}$ được chuyển đổi thành các chuỗi bit bằng hàm BytesToBits khi SHAKE256 được gọi. Chuỗi bit đầu ra của SHAKE256 được chuyển đổi ngược lại thành một mảng byte bằng hàm BitsToBytes.

- Ba hàm băm: Đặc tả của ML-KEM cũng sử dụng ba thể hiện hàm băm ký hiệu là H , J , và G như sau:
Các hàm H và J nhận các đầu vào có độ dài biến đổi và tạo ra các đầu ra kích thước 32-byte. Chúng

lần lượt được ký hiệu là $H: \mathbb{B}^* \rightarrow \mathbb{B}^{32}$ và $J: \mathbb{B}^* \rightarrow \mathbb{B}^{32}$, và phải được triển khai dưới dạng:

$$H(s) = \text{SHA3-256}(s) \quad \text{và} \quad J(s) = \text{SHAKE256}(s, 256),$$

trong đó $s \in \mathbb{B}^*$. Đầu vào $s \in \mathbb{B}^*$ được chuyển đổi thành chuỗi bit bằng hàm BytesToBits khi các hàm SHA3 và SHAKE256 được gọi. Mỗi chuỗi bit đầu ra của SHA3 và SHAKE256 được chuyển đổi thành một mảng byte bằng hàm BitsToBytes.

Hàm G nhận một đầu vào có độ dài biến đổi và tạo ra hai đầu ra kích thước 32-byte. Hàm được ký hiệu là $G: \mathbb{B}^* \rightarrow \mathbb{B}^{32} \times \mathbb{B}^{32}$. Hai đầu ra của G được ký hiệu bởi, ví dụ: $(a, b) \leftarrow G(c)$, trong đó $a, b \in \mathbb{B}^{32}$, $c \in \mathbb{B}^*$, và $G(c) = a \parallel b$. Hàm G phải được triển khai dưới dạng:

$$G(c) = \text{SHA3-512}(c),$$

trong đó đầu vào $c \in \mathbb{B}^*$ được chuyển đổi thành một chuỗi bit bằng hàm BytesToBits khi SHA3-512 được gọi. Chuỗi bit đầu ra của SHA3-512 (gồm hai phân đoạn 256-bit) sẽ được chuyển đổi thành hai mảng byte a và b , mỗi mảng rộng 32 byte, bằng hàm BitsToBytes.

6.3. Các thuật toán nén và giải nén

Trong ML-KEM, tham số $q = 3329$ và độ dài bit của q là 12. Đối với $d < 12$, định nghĩa các hàm:

$$\text{Compress}_d: \mathbb{Z}_q \rightarrow \mathbb{Z}_{2^d}$$

$$x \mapsto \left\lfloor \left(\frac{2^d}{q} \right) \cdot x \right\rfloor \bmod 2^d$$

$$\text{Decompress}_d: \mathbb{Z}_{2^d} \rightarrow \mathbb{Z}_q$$

$$y \mapsto \left\lfloor \left(\frac{q}{2^d} \right) \cdot y \right\rfloor.$$

Lưu ý rằng kiểu dữ liệu đầu vào và đầu ra của các hàm này là các số nguyên đồng dư theo mô-đun của một số nguyên m . Phép chia và phép làm tròn trong quá trình tính toán của các hàm trên được thực hiện trên tập hợp các số hữu tỉ. Không được phép sử dụng các tính toán số dấu phẩy động.

Về mặt bản chất, hàm Compress loại bỏ các bit bậc thấp của đầu vào, và hàm Decompress thêm vào các bit bậc thấp được đặt bằng 0. Các thuật toán này thỏa mãn hai tính chất quan trọng. Thứ nhất, quá trình giải nén rồi nén lại sẽ bảo toàn nguyên vẹn dữ liệu đầu vào, nghĩa là $\text{Compress}_d(\text{Decompress}_d(y)) = y$ với mọi $y \in \mathbb{Z}_{2^d}$ và mọi $d < 12$. Thứ hai, nếu d lớn (tức là gần bằng 12) — đồng nghĩa với số lượng bit bị loại bỏ là nhỏ — thì quá trình nén rồi giải nén lại không làm thay đổi đáng kể giá trị gốc.

6.4. Mã hóa biểu diễn và giải mã biểu diễn

6.4.1. Các tham số

Các thuật toán ByteEncode và ByteDecode được sử dụng để chuyển đổi qua lại giữa các số nguyên theo mô-đun m và các byte. ByteEncode chuyển đổi một mảng gồm 256 số nguyên theo mô-đun m thành một mảng các byte tương ứng. ByteDecode thực hiện phép toán ngược lại, chuyển đổi một mảng các byte thành một mảng gồm các số nguyên theo mô-đun m .

Để thuận tiện cho việc thảo luận tiếp theo, có thể coi ByteDecode và ByteEncode như là các hàm chuyển

đổi qua lại giữa số nguyên và các bit. Phép chuyển đổi giữa bit và byte rất trực quan và được thực hiện bằng cách sử dụng các hàm BitsToBytes và BytesToBits.

Phạm vi giá trị hợp lệ của tham số d là $1 \leq d \leq 12$. Các mảng bit được chia thành các phân đoạn d -bit. Trong trường hợp $1 \leq d \leq 11$, ByteDecode_d chuyển đổi mỗi phân đoạn d -bit của đầu vào thành một số nguyên theo mô-đun 2^d , và ByteEncode_d thực hiện phép toán ngược lại. Trong trường hợp này, phép chuyển đổi là song ánh. Trường hợp $d = 12$ được xử lý khác biệt. Trong trường hợp này, ByteEncode_{12} nhận đầu vào là các số nguyên theo mô-đun q , và ByteDecode_{12} tạo ra đầu ra là các số nguyên theo mô-đun q . ByteDecode_{12} chuyển đổi mỗi phân đoạn 12-bit của đầu vào thành một số nguyên theo mô-đun $2^{12} = 4096$, sau đó rút gọn kết quả theo mô-đun q . Đây không còn là một phép toán song ánh nữa. Thực tế, một số phân đoạn 12-bit có thể tương ứng với một số nguyên lớn hơn $q - 1 = 3328$ nhưng nhỏ hơn 4096; tuy nhiên, các trường hợp này không xảy ra đối với các mảng được tạo ra bởi hàm ByteEncode_{12} .

6.4.2. Mã hóa biểu diễn

Thuật toán $\text{ByteEncode}_d(F)$ chuyển một mảng các số nguyên d -bit thành một mảng byte, với $1 \leq d \leq 12$. Thuật toán nhận đầu vào là một mảng số nguyên $F \in \mathbb{Z}_m^{256}$, trong đó $m = 2^d$ nếu $d < 12$ và $m = q$ nếu $d = 12$, và đưa ra một mảng byte $B \in \mathbb{B}^{32d}$ như sau:

1. For $i = 0$ to 255 do:
2. $a \leftarrow F[i]$
3. For $j = 0$ to $d - 1$:
4. $b[i \cdot d + j] \leftarrow a \bmod 2$, trong đó b là một chuỗi bit độ dài $256d$
5. $a \leftarrow (a - b[i \cdot d + j])/2$
6. End for
7. End for
8. $B \leftarrow \text{BitsToBytes}(b)$
9. Return B

6.4.3. Giải mã biểu diễn

Thuật toán $\text{ByteDecode}_d(B)$ chuyển một mảng byte thành một mảng F gồm các số nguyên d -bit, với $1 \leq d \leq 12$. Thuật toán nhận đầu vào là một mảng byte $B \in \mathbb{B}^{32d}$ và đưa ra một mảng số nguyên $F \in \mathbb{Z}_m^{256}$, trong đó $m = 2^d$ nếu $d < 12$ và $m = q$ nếu $d = 12$, như sau:

1. $b \leftarrow \text{BytesToBits}(B)$
2. For $i = 0$ to 255 do:
3. $F[i] \leftarrow (\sum_{j=0}^{d-1} b[i \cdot d + j] \cdot 2^j) \bmod m$
4. End for
5. Return F

6.5. Các thuật toán lấy mẫu

6.5.1. Tổng quan

Các thuật toán của ML-KEM yêu cầu hai thuật toán con lấy mẫu được quy định cụ thể trong thuật toán $\text{SampleNTT}(B)$ và $\text{SamplePolyCBD}_\eta(B)$. Cả hai thuật toán này đều có thể được sử dụng để chuyển đổi một luồng các byte ngẫu nhiên phân phối đều thành một mẫu theo phân phối mong muốn nào đó. Trong tiêu chuẩn này, các thuật toán trên được gọi với đầu vào là một luồng các byte giả ngẫu nhiên. Do đó, kết quả đầu ra sẽ là một mẫu theo phân phối không thể phân biệt được về mặt tính toán so với phân phối mong muốn.

Thuật toán SampleNTT chuyển đổi một luồng các byte thành một đa thức trong miền NTT. Nếu luồng đầu vào gồm các byte ngẫu nhiên phân phối đều, thì kết quả thu được là ngẫu nhiên phân phối đều trên T_q . Đầu ra là một mảng thuộc $\mathbb{Z}_q^{256}$ chứa các hệ số của phần tử được lấy mẫu thuộc T_q (xem mục 6.5.2).

6.5.2. $\text{SampleNTT}(B)$

Nếu đầu vào là một luồng các byte ngẫu nhiên phân phối đều, đầu ra của $\text{SampleNTT}(B)$ là một phần tử ngẫu nhiên phân phối đều thuộc T_q . Thuật toán nhận đầu vào là luồng byte $B \in \mathbb{B}^*$ và đưa ra một mảng $\hat{a} \in \mathbb{Z}_q^{256}$ như sau:

1. $i \leftarrow 0$
2. $j \leftarrow 0$
3. While $j < 256$ do:
 4. $d_1 \leftarrow B[i] + 256 \cdot (B[i + 1] \bmod 16)$
 5. $d_2 \leftarrow \lfloor B[i + 1]/16 \rfloor + 16 \cdot B[i + 2]$
 6. If $d_1 < q$ then $\hat{a}[j] \leftarrow d_1$ and $j \leftarrow j + 1$
 7. If $d_2 < q$ and $j < 256$ then $\hat{a}[j] \leftarrow d_2$ and $j \leftarrow j + 1$
 8. $i \leftarrow i + 3$
9. End while
10. Return $\hat{a}$

6.5.3. $\text{SamplePolyCBD}_\eta(B)$

ML-KEM sử dụng một phân phối đặc biệt $\mathcal{D}_\eta(R_q)$ của các đa thức trong R_q với các hệ số nhỏ. Các đa thức như vậy đôi khi được gọi là “lỗi” hoặc “nhiều”. Phân phối này được tham số hóa bởi một số nguyên η . Để lấy mẫu một đa thức từ $\mathcal{D}_\eta(R_q)$, mỗi hệ số của nó được lấy mẫu một cách độc lập từ một phân phối nhị thức trung tâm nhất định trên $\mathbb{Z}_q$. Thuật toán SamplePolyCBD lấy mẫu mảng hệ số của một đa thức $f \in R_q$ theo phân phối $\mathcal{D}_\eta(R_q)$, với điều kiện đầu vào của nó là một mảng các byte ngẫu nhiên phân phối đều. Thuật toán nhận đầu vào là một mảng byte $B \in \mathbb{B}^{64\eta}$ và đưa ra một mảng $f \in \mathbb{Z}_q^{256}$ như sau:

1. $b \leftarrow \text{BytesToBits}(B)$
2. For $i = 0$ to 255 do:

3. $x \leftarrow \sum_{j=0}^{\eta-1} b[2i\eta + j]$
4. $y \leftarrow \sum_{j=0}^{\eta-1} b[2i\eta + \eta + j]$
5. $f[i] \leftarrow (x - y) \bmod q$
6. End for
7. Return f

6.6. Phép biến đổi lý thuyết số

6.6.1. Tổng quan

Phép biến đổi lý thuyết số (NTT) có thể được coi là một phiên bản đặc biệt, chính xác của phép biến đổi Fourier rời rạc. Trong trường hợp của ML-KEM, NTT được sử dụng để nâng cao hiệu năng của phép nhân trong vành R_q . Nhắc lại rằng R_q là vành đa thức $\mathbb{Z}_q[X]/(X^n + 1)$, bao gồm các đa thức có dạng $f = f_0 + f_1X + \dots + f_{255}X^{255}$, trong đó $f_j \in \mathbb{Z}_q$ với mọi j , đi kèm với các phép toán số học theo mô-đun $X^n + 1$.

Vành R_q đẳng cấu tự nhiên với một vành khác, ký hiệu là T_q . NTT là một phép đẳng cấu có hiệu năng tính toán cao giữa hai vành này. Khi nhận đầu vào là một đa thức $f \in R_q$, thuật toán NTT đưa ra một phần tử $\hat{f} := \text{NTT}(f) \in T_q$ trong đó $\hat{f}$ được gọi là “biểu diễn NTT” của f . Tính chất đẳng cấu này suy ra rằng $f \times_{R_q} g = \text{NTT}^{-1}(\hat{f} \times_{T_q} \hat{g})$ trong đó $\times_{R_q}$ và $\times_{T_q}$ lần lượt biểu thị phép nhân trong vành R_q và T_q . Hơn nữa, vì T_q là tích của 128 vành thành phần (mỗi vành bao gồm các đa thức có bậc bé hơn 2), phép toán $\times_{T_q}$ có hiệu năng xử lý cao hơn rất nhiều so với phép toán $\times_{R_q}$. Vì những lý do này, NTT được xem là một phần cấu thành của ML-KEM chứ không đơn thuần là một giải pháp tối ưu.

Do hai vành R_q và T_q đều có cấu trúc không gian vector trên trường $\mathbb{Z}_q$, kiểu dữ liệu trừu tượng tự nhiên nhất để đại diện cho các phần tử của cả hai vành này là $\mathbb{Z}_q^n$. Vì vậy, cấu trúc dữ liệu được lựa chọn cho đầu vào và đầu ra của hai hàm NTT và NTT^{-1} là các mảng có độ dài n gồm các số nguyên theo mô-đun q ; các mảng này được hiểu là đại diện cho các phần tử thuộc T_q hoặc R_q (xem mục 6.6.2 và 6.6.3). Các thuật toán trong mục 6.6.2 và 6.6.3 thể hiện một phương thức tính toán tối ưu cho NTT và NTT^{-1} . Tuy nhiên, để làm rõ sự khác biệt giữa các đối tượng đại số trước và sau khi biến đổi, các thuật toán được viết kèm theo mô tả đầu vào và đầu ra một cách tường minh.

Trong ML-KEM, q là số nguyên tố $3329 = 2^8 \cdot 13 + 1$ và $n = 256$. Trong $\mathbb{Z}_q$ có đúng 128 căn nguyên thủy bậc 256 của đơn vị, và không tồn tại căn nguyên thủy bậc 512 của đơn vị. Lưu ý rằng $\zeta = 17 \in \mathbb{Z}_q$ là một căn nguyên thủy bậc 256 của đơn vị theo mô-đun q . Do đó, ta có $\zeta^{128} \equiv -1$.

Định nghĩa $\text{BitRev}_7(i)$ là số nguyên thu được bằng cách đảo ngược chuỗi bit của giá trị 7-bit không dấu tương ứng với số nguyên đầu vào $i \in \{0, 1, \dots, 127\}$.

Đa thức $X^{256} + 1$ có thể phân tích thành tích của 128 đa thức bậc 2 theo mô-đun q như sau:

$$X^{256} + 1 = \prod_{k=0}^{127} (X^2 - \zeta^{2\text{BitRev}_7(k)+1}).$$

Do đó, vành $R_q := \mathbb{Z}_q[X]/(X^{256} + 1)$ đẳng cấu với tổng trực tiếp của 128 trường mở rộng bậc hai của $\mathbb{Z}_q$, ký hiệu là T_q . Cụ thể, vành này có cấu trúc như sau:

$$T_q := \bigoplus_{k=0}^{127} \mathbb{Z}_q[X]/(X^2 - \zeta^{2\text{BitRev}_7(k)+1}).$$

Vì vậy, biểu diễn NTT $\hat{f} \in T_q$ của một đa thức $f \in R_q$ chính là một vector gồm các phần dư đa thức tương ứng có bậc không quá một:

$$\hat{f} := \left(f \bmod (X^2 - \zeta^{2\text{BitRev}_7(0)+1}), \dots, f \bmod (X^2 - \zeta^{2\text{BitRev}_7(127)+1}) \right).$$

Trong các thuật toán được mô tả bên dưới, $\hat{f}$ được lưu trữ dưới dạng một mảng gồm 256 số nguyên theo mô-đun q . Cụ thể,

$$f \bmod (X^2 - \zeta^{2\text{BitRev}_7(i)+1}) = \hat{f}[2i] + \hat{f}[2i+1]X$$

với i chạy từ 0 đến 127.

6.6.2. NTT(f)

Thuật toán NTT(f) tính biểu diễn NTT $\hat{f}$ của một đa thức cho trước $f \in R_q$. Thuật toán nhận đầu vào là một mảng $f \in \mathbb{Z}_q^{256}$ và đưa ra một mảng $\hat{f} \in \mathbb{Z}_q^{256}$, như sau:

1. $\hat{f} \leftarrow f$
2. $k \leftarrow 1$
3. For $len = 128$ to 2, giảm giá trị len bằng cách đặt $len \leftarrow len/2$ do:
4. For $start = 0$ to 255, tăng giá trị $start$ bằng cách đặt $start \leftarrow start + 2 \cdot len$ do:
5. $zeta \leftarrow \zeta^{\text{BitRev}_7(k)} \bmod q$
6. $k \leftarrow k + 1$
7. For $j = start$ to $(start + len - 1)$ do:
8. $t \leftarrow zeta \cdot \hat{f}[j + len] \bmod q$
9. $\hat{f}[j + len] \leftarrow (\hat{f}[j] - t) \bmod q$
10. $\hat{f}[j] \leftarrow (\hat{f}[j] + t) \bmod q$
11. End for
12. End for
13. End for
14. Return $\hat{f}$

6.6.3. NTT⁻¹($\hat{f}$)

Thuật toán NTT⁻¹($\hat{f}$) tính đa thức $f \in R_q$ tương ứng với biểu diễn NTT cho trước $\hat{f} \in T_q$. Thuật toán nhận đầu vào là một mảng $\hat{f} \in \mathbb{Z}_q^{256}$ và đưa ra một mảng $f \in \mathbb{Z}_q^{256}$ như sau:

1. $f \leftarrow \hat{f}$
2. $k \leftarrow 127$
3. For $len = 2$ to 128 , tăng giá trị len bằng cách đặt $len \leftarrow 2 \cdot len$ do:
4. For $start = 0$ to 255 , tăng giá trị $start$ bằng cách đặt $start \leftarrow start + 2 \cdot len$ do:
5. $zeta \leftarrow \zeta^{\text{BitRev}_7(k)} \bmod q$
6. $k \leftarrow k - 1$
7. For $j = start$ to $(start + len - 1)$ do:
8. $t \leftarrow f[j] \bmod q$
9. $f[j] \leftarrow (t + f[j + len]) \bmod q$
10. $f[j + len] \leftarrow zeta \cdot (f[j + len] - t) \bmod q$
11. End for
12. End for
13. End for
14. $f \leftarrow f \cdot 3303 \bmod q$
15. Return f

6.7. Phép nhân trong miền NTT

6.7.1. Tổng quan

Như đã thảo luận trong mục 5.3, phép cộng và phép nhân vô hướng của các phần tử trong vành T_q rất trực quan: chúng có thể được thực hiện bằng các phép toán số học tương ứng theo từng tọa độ trên các mảng hệ số. Mục này mô tả cách thức thực hiện phép nhân trong vành T_q .

Từ mô tả trong mục 6.6.1, nhắc lại rằng $\hat{f} \in T_q$ là một vector gồm 128 phần dư bậc không quá một theo mô-đun của các đa thức bậc hai. Về mặt đại số, phép nhân trong vành T_q bao gồm các phép nhân độc lập trong từng tọa độ của 128 tọa độ theo mô-đun bậc hai của tọa độ đó. Cụ thể, tọa độ thứ i trong T_q của tích $\hat{h} = \hat{f} \times_{T_q} \hat{g}$ được xác định thông qua phép tính:

$$\hat{h}[2i] + \hat{h}[2i + 1]X = (\hat{f}[2i] + \hat{f}[2i + 1]X)(\hat{g}[2i] + \hat{g}[2i + 1]X) \bmod (X^2 - \zeta^{2\text{BitRev}_7(i)+1}).$$

Do đó, hoàn toàn có thể tính được tích của hai phần tử trong T_q bằng cách sử dụng thuật toán MultiplyNTTs. Lưu ý rằng MultiplyNTTs sử dụng BaseCaseMultiply như một thuật toán con. Như đã thảo luận trong mục 5.3, MultiplyNTTs cho phép thực hiện các phép tính đại số tuyến tính với ma trận và vector có các phần tử nằm trong vành T_q .

6.7.2. MultiplyNTTs($\hat{f}, \hat{g}$)

Thuật toán MultiplyNTTs($\hat{f}, \hat{g}$) tính toán tích (trong vành T_q) của hai biểu diễn NTT. Thuật toán nhận đầu vào là hai mảng $\hat{f} \in \mathbb{Z}_q^{256}$ và $\hat{g} \in \mathbb{Z}_q^{256}$, đưa ra một mảng $\hat{h} \in \mathbb{Z}_q^{256}$ như sau:

1. For $i = 0$ to 127 do:
2. $(\hat{h}[2i], \hat{h}[2i + 1]) \leftarrow \text{BaseCaseMultiply}(\hat{f}[2i], \hat{f}[2i + 1], \hat{g}[2i], \hat{g}[2i + 1], \zeta^{2\text{BitRev}_7(i)+1})$
3. End for
4. Return $\hat{h}$

6.7.3. BaseCaseMultiply($a_0, a_1, b_0, b_1, \gamma$)

Thuật toán BaseCaseMultiply($a_0, a_1, b_0, b_1, \gamma$) tính toán tích của hai đa thức $a_0 + a_1X$ và $b_0 + b_1X$ theo mô-đun $X^2 - \gamma$. Thuật toán nhận đầu vào là $a_0, a_1, b_0, b_1 \in \mathbb{Z}_q$ và $\gamma \in \mathbb{Z}_q$, đưa ra hai giá trị $c_0, c_1 \in \mathbb{Z}_q$ tương ứng với đa thức kết quả $c_0 + c_1X$ như sau:

1. $c_0 \leftarrow (a_0 \cdot b_0 + a_1 \cdot b_1 \cdot \gamma) \bmod q$
2. $c_1 \leftarrow (a_0 \cdot b_1 + a_1 \cdot b_0) \bmod q$
3. Return c_0, c_1

7. Lược đồ thành phần K-PKE

7.1. Tổng quan

K-PKE đóng vai trò là một tập hợp các chương trình con được sử dụng bên trong các thuật toán của ML-KEM. K-PKE bao gồm ba thuật toán:

1. Sinh khóa (K-PKE.KeyGen);
2. Mã hóa (K-PKE.Encrypt);
3. Giải mã (K-PKE.Decrypt).

Khi K-PKE được triển khai như một phần của ML-KEM, nó sẽ kế thừa toàn bộ bộ tham số được lựa chọn cho ML-KEM đó. Mỗi bộ tham số quy định các giá trị số cụ thể cho từng tham số. Trong khi các hằng số n luôn bằng 256, q luôn bằng 3329 và $\eta_1 = \eta_2$ luôn cố định bằng 2, thì giá trị của các tham số còn lại sẽ thay đổi tùy thuộc vào từng bộ tham số cụ thể. Chi tiết về các tham số đơn lẻ và các bộ tham số được mô tả chi tiết trong mục 9.

Mỗi thuật toán của K-PKE được kèm theo một mô tả ngắn gọn, không chính thức dưới dạng văn bản. Để đơn giản hóa, phần diễn giải này được viết dưới dạng các vector và ma trận mà các phần tử của chúng thuộc vành R_q . Trong thuật toán triển khai thực tế, hầu hết các phép tính toán đều diễn ra trong miền NTT nhằm tối ưu hóa hiệu năng của các phép nhân. Khi đó, các vector và ma trận liên quan sẽ chứa các phần tử thuộc vành T_q . Các phép toán đại số tuyến tính với các vector và ma trận miền NTT này được thực hiện chính xác theo quy định tại mục 5.3 và 5.4. Các khóa mã hóa và khóa giải mã của K-PKE cũng được lưu trữ trực tiếp dưới dạng biểu diễn NTT.

Lược đồ mã hóa khóa công khai K-PKE được mô tả trong tiêu chuẩn này không được phép sử dụng như một lược đồ mật mã độc lập. Thay vào đó, các thuật toán cấu thành nên K-PKE chỉ có thể được sử dụng làm các hàm con trong cấu trúc thuật toán của ML-KEM. Cụ thể, các thuật toán K-PKE.KeyGen, K-PKE.Encrypt và K-PKE.Decrypt không được sử dụng như một lược đồ mã hóa khóa công khai.

7.2. Sinh khóa K-PKE

7.2.1. Tổng quan

Từ góc nhìn thông thường của mã hóa khóa công khai, khóa mã hóa có thể được công khai, trong khi khóa giải mã và nguồn dữ liệu ngẫu nhiên bắt buộc phải được giữ bí mật. Điều này hoàn toàn đúng trong bối cảnh của tiêu chuẩn này. Khóa mã hóa của K-PKE đóng vai trò là khóa đóng gói của ML-KEM (xem mục ML-KEM.KeyGen tại 8.1), do đó nó có thể được công khai; ngược lại, khóa giải mã và nguồn ngẫu nhiên của thuật toán K-PKE.KeyGen phải được giữ bí mật tuyệt đối, vì chúng được sử dụng để thực hiện quá trình giải gói khóa trong ML-KEM.

Khóa giải mã của K-PKE.KeyGen là một vector s có độ dài k gồm các phần tử thuộc vành R_q (tức là $s \in R_q^k$). Nói một cách dễ hiểu, s đại diện cho một tập hợp các biến bí mật, trong đó khóa mã hóa là các phương trình tuyến tính “chứa nhiều” ($A, As + e$) chứa các biến bí mật s . Các hàng của ma trận A tạo thành các hệ số của phương trình. Ma trận này được khởi tạo giả ngẫu nhiên bằng hàm SHAKE128, và chỉ có mầm của nó được lưu lại trong khóa mã hóa. Vector bí mật s và vector “nhiều” e được lấy mẫu từ phân phối nhị thức trung tâm bằng cách sử dụng nguồn ngẫu nhiên được mở rộng từ một mầm thông qua hàm PRF. Sau khi A, s và e được khởi tạo, phép toán tính toán thành phần cuối cùng $t = As + e$ của khóa mã hóa sẽ được thực hiện.

Trong thuật toán K-PKE.KeyGen, việc lựa chọn bộ tham số sẽ ảnh hưởng đến độ dài của vector bí mật s (thông qua tham số k), và kéo theo đó là kích thước của vector nhiều e cùng ma trận giả ngẫu nhiên A . Việc lựa chọn bộ tham số này cũng quyết định dạng phân phối nhiều (thông qua tham số η_1) được áp dụng để lấy mẫu cho các phần tử của s và e .

7.2.2. K-PKE.KeyGen(d)

Thuật toán ngẫu nhiên K-PKE.KeyGen(d) sinh một khóa mã hóa và một khóa giải mã tương ứng. Thuật toán nhận đầu vào là một mảng ngẫu nhiên $d \xleftarrow{\$} \mathbb{B}^{32}$ và đưa ra một khóa mã hóa $ek_{PKE} \in \mathbb{B}^{384k+32}$ và một khóa giải mã $dk_{PKE} \in \mathbb{B}^{384k}$ như sau:

1. $(\rho, \sigma) \leftarrow G(d \parallel k)$, trong đó k được biểu diễn bằng 1 byte; ρ và σ là hai mầm giả ngẫu nhiên 32-byte.
2. $N \leftarrow 0$
3. For $i = 0$ to $k - 1$ do:
4. For $j = 0$ to $k - 1$ do:
5. $\hat{A}[i, j] \leftarrow \text{SampleNTT}(\text{SHAKE128}(\rho \parallel j \parallel i))$, trong đó đầu vào của hàm SHAKE128 là một chuỗi 34-byte thu được bằng cách nối ρ, j và i (trong đó j và i đều được biểu diễn như các phần tử thuộc $\mathbb{B}$).
6. End for
7. End for
8. For $i = 0$ to $k - 1$ do:
9. $s[i] \leftarrow \text{SamplePolyCBD}_{\eta_1}(\text{PRF}_{\eta_1}(\sigma, N))$

10. $N \leftarrow N + 1$
11. End for
12. For $i = 0$ to $k - 1$ do:
13. $e[i] \leftarrow \text{SamplePolyCBD}_{\eta_1}(\text{PRF}_{\eta_1}(\sigma, N))$
14. $N \leftarrow N + 1$
15. End for
16. $\hat{s} \leftarrow \text{NTT}(s)$, trong đó hàm NTT được chạy k lần (mỗi lần cho một tọa độ của s).
17. $\hat{e} \leftarrow \text{NTT}(e)$, trong đó hàm NTT được chạy k lần (mỗi lần cho một tọa độ của e).
18. Tính $\hat{t} \leftarrow \hat{A} \circ \hat{s} + \hat{e}$, tạo thành một hệ tuyến tính chứa nhiều trong miền NTT.
19. $ek_{PKE} \leftarrow \text{ByteEncode}_{12}(\hat{t}) \parallel \rho$, trong đó hàm ByteEncode_{12} được chạy k lần và mã cho $\hat{A}$ được nối thêm vào cuối.
20. $dk_{PKE} \leftarrow \text{ByteEncode}_{12}(\hat{s})$, trong đó hàm ByteEncode_{12} được chạy k lần.
21. Return (ek_{PKE}, dk_{PKE}) .

7.3 Mã hóa K-PKE

7.3.1. Tổng quan

Thuật toán K-PKE.Encrypt bắt đầu bằng việc trích xuất vector t và mã từ khóa mã hóa. Mã này sau đó được mở rộng để sinh lại ma trận A hoàn toàn giống với cách thức đã thực hiện trong thuật toán K-PKE.KeyGen. Nếu t và A được trích xuất chính xác từ một khóa mã hóa do K-PKE.KeyGen tạo ra, chúng sẽ trùng khớp hoàn toàn với các giá trị tương ứng trong K-PKE.KeyGen. Như đã thảo luận ở phần sinh khóa, cặp $(A, t = As + e)$ có thể được coi là một hệ phương trình tuyến tính chứa nhiều theo các biến bí mật s . Một phương trình tuyến tính chứa nhiều bổ sung có thể được sinh theo cùng các biến bí mật — mà không cần biết s — bằng cách chọn một tổ hợp tuyến tính ngẫu nhiên của các phương trình chứa nhiều sẵn có trong hệ (A, t) . Thông tin sau đó có thể được mã hóa trực tiếp vào “hạng tử tự do” (tức là thành phần được tạo ra từ tổ hợp tuyến tính của các phần tử trong t) của phương trình tổ hợp mới này. Chuỗi thông tin mã hóa đó hoàn toàn có thể giải mã được bởi một bên nắm giữ bí mật s . Ví dụ, ta có thể mã hóa một bit đơn lẻ bằng cách quyết định xem có thay đổi đáng kể giá trị của hạng tử tự do hay không, qua đó tạo ra một phương trình gần như chính xác (tương ứng với giá trị bit được giải mã là 0) hoặc một phương trình sai lệch rất lớn (tương ứng với giá trị bit được giải mã là 1). Trong trường hợp của K-PKE, hạng tử tự do là một đa thức có 256 hệ số, cho phép ta mã hóa được nhiều thông tin hơn: một bit dữ liệu trên mỗi hệ số đa thức. Để đạt được điều này, K-PKE.Encrypt tiến hành khởi tạo một vector $r \in R_q^k$ cùng các thành phần nhiễu $e_1 \in R_q^k$ và $e_2 \in R_q$. Tất cả các giá trị này đều được lấy mẫu từ phân phối nhị thức tập trung (CBD) bằng cách sử dụng nguồn dữ liệu giả ngẫu nhiên được mở rộng (qua hàm PRF) từ chuỗi ngẫu nhiên đầu vào r . “Hệ phương trình chứa nhiều mới” sau đó được tính toán, xác định cụ thể bởi công thức: $u \leftarrow A^T r + e_1$. Tiếp theo, v thu được bằng cách cộng một bản mã hóa phù hợp μ của thông điệp đầu vào m vào biểu thức $t^T r + e_2$. Cuối cùng, cặp (u, v) được nén lại, tuần tự hóa thành một mảng byte và đưa ra dưới dạng bản mã.

7.3.2. K-PKE.Encrypt(ek_{PKE}, m, r)

Thuật toán này sử dụng một khóa mã hóa ek_{PKE} để mã hóa bản rõ m với nguồn ngẫu nhiên mã hóa r . Lưu ý rằng vì ML-KEM cần gọi hàm K-PKE.Encrypt với một lựa chọn nguồn ngẫu nhiên cụ thể (xem mục 8.3 để biết chi tiết), chuỗi byte r sẽ được lấy mẫu bên ngoài thuật toán này. Do đó, thuật toán K-PKE.Encrypt là tất định.

Thuật toán K-PKE.Encrypt nhận đầu vào là một khóa mã hóa $ek_{PKE} \in \mathbb{B}^{384k+32}$, một thông điệp bản rõ $m \in \mathbb{B}^{32}$, và một chuỗi byte $r \in \mathbb{B}^{32}$, đưa ra một bản mã $c \in \mathbb{B}^{32(d_u k + d_v)}$ như sau:

1. $N \leftarrow 0$
2. $\hat{t} \leftarrow \text{ByteDecode}_{12}(ek_{PKE}[0:384k])$
3. $\rho \leftarrow ek_{PKE}[384k:384k+32]$
4. For $i = 0$ to $k - 1$ do:
5. For $j = 0$ to $k - 1$ do
6. $\hat{A}[i, j] \leftarrow \text{SampleNTT}(\text{SHAKE128}(\rho \parallel j \parallel i))$
7. End for
8. End for
9. For $i = 0$ to $k - 1$ do
10. $r[i] \leftarrow \text{SamplePolyCBD}_{\eta_1}(\text{PRF}_{\eta_1}(r, N))$
11. $N \leftarrow N + 1$
12. End for
13. For $i = 0$ to $k - 1$ do
14. $e_1[i] \leftarrow \text{SamplePolyCBD}_{\eta_2}(\text{PRF}_{\eta_2}(r, N))$
15. $N \leftarrow N + 1$
16. End for
17. $e_2 \leftarrow \text{SamplePolyCBD}_{\eta_2}(\text{PRF}_{\eta_2}(r, N))$
18. $\hat{r} \leftarrow \text{NTT}(r)$
19. $u \leftarrow \text{NTT}^{-1}(\hat{A}^T \circ \hat{r}) + e_1$
20. $\mu \leftarrow \text{Decompress}_1(\text{ByteDecode}_1(m))$
21. $v \leftarrow \text{NTT}^{-1}(\hat{t}^T \circ \hat{r}) + e_2 + \mu$
22. $c_1 \leftarrow \text{ByteEncode}_{d_u}(\text{Compress}_{d_u}(u))$
23. $c_2 \leftarrow \text{ByteEncode}_{d_v}(\text{Compress}_{d_v}(v))$
24. Return $c \leftarrow (c_1 \parallel c_2)$

7.4 Giải mã K-PKE

7.4.1 Tổng quan

Thuật toán giải mã K-PKE.Decrypt của K-PKE nhận đầu vào là một khóa giải mã dk_{PKE} và một bản mã c , sau đó đưa ra một bản rõ m .

Thuật toán K-PKE.Decrypt bắt đầu bằng việc tính toán “phương trình nhiễu” (u, v) ẩn trong bản mã c , như đã được thảo luận trong phần mô tả về K-PKE.Encrypt. Tại đây, u có thể được coi là hệ số của phương trình và v là hạng tử tự do. Nhắc lại rằng khóa giải mã dk_{PKE} chứa vectơ của các biến bí mật s . Do đó, thuật toán giải mã có thể sử dụng khóa giải mã để tính toán hạng tử tự do thực tế $v' = s^T u$ và sau đó tính $v - v'$. Thuật toán giải mã kết thúc bằng việc giải mã biểu diễn thông điệp bản rõ m từ $v - v'$ và đưa ra m .

Đặc tả thuật toán được cung cấp trong 7.4.2.

7.4.2 K-PKE.Decrypt(dk_{PKE}, c)

K-PKE.Decrypt(dk_{PKE}, c) sử dụng khóa giải mã để giải mã một bản mã. Thuật toán nhận đầu vào là một khóa giải mã $dk_{PKE} \in \mathbb{B}^{384k}$ và một bản mã $c \in \mathbb{B}^{32(d_u k + d_v)}$, và đưa ra một thông điệp $m \in \mathbb{B}^{32}$.

1. $c_1 \leftarrow c[0:32d_u k]$
2. $c_2 \leftarrow c[32d_u k:32(d_u k + d_v)]$
3. $u' \leftarrow \text{Decompress}_{d_u}(\text{ByteDecode}_{d_u}(c_1))$
4. $v' \leftarrow \text{Decompress}_{d_v}(\text{ByteDecode}_{d_v}(c_2))$
5. $\hat{s} \leftarrow \text{ByteDecode}_{12}(dk_{PKE})$
6. $w \leftarrow v' - \text{NTT}^{-1}(\hat{s}^T \circ \text{NTT}(u'))$
7. $m \leftarrow \text{ByteEncode}_1(\text{Compress}_1(w))$
8. Return m

8. Cơ chế đóng gói khóa

8.1. Tổng quan

Hai bộ tham số được định nghĩa cho ML-KEM, cho phép cân bằng giữa độ an toàn và hiệu năng. Hai bộ tham số khả thi được gọi là ML-KEM-768 và ML-KEM-1024, và được mô tả chi tiết trong Bảng 1. Mỗi bộ tham số gán các giá trị số cụ thể cho từng tham số riêng lẻ $n, q, k, \eta_1, \eta_2, d_u$ và d_v . Trong khi n luôn là 256, q luôn là 3329 và $\eta_1 = \eta_2$ luôn là 2, các tham số còn lại thay đổi tùy thuộc vào từng bộ tham số. Người triển khai ứng dụng phải đảm bảo rằng ba thuật toán của ML-KEM liệt kê ở trên chỉ được gọi với một bộ tham số hợp lệ, và bộ tham số này được lựa chọn một cách phù hợp cho ứng dụng mong muốn.

8.2. Sinh khóa

8.2.1. Tổng quan

Thuật toán sinh khóa ngẫu nhiên ML-KEM.KeyGen của ML-KEM không nhận đầu vào và tạo ra một khóa đóng gói và một khóa giải gói. Trong khi khóa đóng gói có thể được công khai, khóa giải gói phải được

giữ bí mật.

Chương trình con cốt lõi của ML-KEM.KeyGen là thuật toán sinh khóa của K-PKE. Khóa đóng gói ML-KEM đơn giản là khóa mã hóa của K-PKE. Khóa giải gói ML-KEM bao gồm khóa giải mã của K-PKE, khóa đóng gói, một giá trị băm của khóa đóng gói, và một giá trị giả ngẫu nhiên 32 byte. Giá trị giả ngẫu nhiên này được sử dụng trong cơ chế “tù chối ngầm” của thuật toán giải gói khóa.

8.2.2. ML-KEM.KeyGen()

ML-KEM.KeyGen() sinh ra một khóa đóng gói và một khóa giải gói tương ứng. Thuật toán ngẫu nhiên này không nhận đầu vào, và đưa ra một khóa đóng gói $ek \in \mathbb{B}^{384k+32}$ và một khóa giải gói $dk \in \mathbb{B}^{768k+96}$.

1. $d \xleftarrow{\$} \mathbb{B}^{32}$
2. $z \xleftarrow{\$} \mathbb{B}^{32}$
3. $(ek_{PKE}, dk_{PKE}) \leftarrow \text{K-PKE.KeyGen}(d)$
4. $ek \leftarrow ek_{PKE}$
5. $dk \leftarrow (dk_{PKE} \parallel ek \parallel H(ek) \parallel z)$
6. Return (ek, dk)

8.3. Đóng gói khóa

Hàm đóng gói khóa ML-KEM.Encaps(ek) sử dụng khóa đóng gói để sinh ra một khóa bí mật được chia sẻ và một bản mã liên kết. Hàm nhận đầu vào là một khóa đóng gói $ek \in \mathbb{B}^{384k+32}$ và đưa ra một khóa bí mật được chia sẻ $K \in \mathbb{B}^{32}$ và một bản mã $c \in \mathbb{B}^{32(d_u k + d_v)}$.

1. $m \xleftarrow{\$} \mathbb{B}^{32}$
2. $(K, r) \leftarrow G(m \parallel H(ek))$
3. $c \leftarrow \text{K-PKE.Encrypt}(ek, m, r)$
4. Return (K, c) .

8.4. Giải gói khóa

Hàm giải gói khóa ML-KEM.Decaps(c, dk) sử dụng khóa giải gói để đưa ra một khóa bí mật được chia sẻ từ một bản mã. Hàm nhận đầu vào là một bản mã $c \in \mathbb{B}^{32(d_u k + d_v)}$ và một khóa giải gói $dk \in \mathbb{B}^{768k+96}$, sau đó đưa ra một khóa bí mật được chia sẻ $K \in \mathbb{B}^{32}$.

1. $dk_{PKE} \leftarrow dk[0:384k]$
2. $ek_{PKE} \leftarrow dk[384k:768k+32]$
3. $h \leftarrow dk[768k+32:768k+64]$
4. $z \leftarrow dk[768k+64:768k+96]$
5. $m' \leftarrow \text{K-PKE.Decrypt}(dk_{PKE}, c)$
6. $(K', r') \leftarrow G(m' \parallel h)$

7. $\bar{K} \leftarrow J(z \parallel c)$
8. $c' \leftarrow \text{K-PKE.Encrypt}(ek_{PKE}, m', r')$
9. If $c \neq c'$ then
10. $K' \leftarrow \bar{K}$
11. End if
12. Return K' .

9. Các bộ tham số

ML-KEM được trang bị hai bộ tham số. Mỗi bộ trong hai bộ tham số bao gồm năm tham số riêng lẻ: k, η_1, η_2, d_u , và d_v . Dưới đây là mô tả ngắn gọn và không chính thức về vai trò của các tham số này trong các thuật toán của K-PKE (và do đó là trong ML-KEM). Xem 7 để biết thêm chi tiết.

- Tham số k là bắt buộc để xác định các vector s và e trong K-PKE.KeyGen, cũng như ma trận $\hat{A}$ và các vector r, e_1 , và đa thức e_2 trong K-PKE.Encrypt.
- Tham số η_1 là bắt buộc để xác định phân phối dùng cho việc sinh các vector s và e trong K-PKE.KeyGen và vector r trong K-PKE.Encrypt.
- Tham số η_2 là bắt buộc để xác định phân phối dùng cho việc sinh các vector e_1 và đa thức e_2 trong K-PKE.Encrypt.
- Các tham số d_u và d_v đóng vai trò là các tham số và đầu vào cho các hàm Compress, Decompress, ByteEncode, và ByteDecode trong K-PKE.Encrypt và K-PKE.Decrypt.

ML-KEM sử dụng các bộ tham số được quy định trong Bảng 1. Mỗi bộ tham số cũng quy định một mức độ an toàn đối với việc sinh số ngẫu nhiên (xem ISO/IEC 18031 để biết thêm thông tin). Xem Phụ lục A để biết các tài liệu tham khảo mô tả các đặc tính bổ sung của các bộ tham số này, bao gồm cả các đặc tính an toàn.

Bảng 1– Các bộ tham số cho ML-KEM

	k	η_1	η_2	d_u	d_v	Độ mạnh RBG (bit)
ML-KEM-768	3	2	2	10	4	192
ML-KEM-1024	4	2	2	11	5	256

Kích thước của khóa, bản mã và khóa bí mật được chia sẻ được tóm tắt trong Bảng 2.

Bảng 2 – Kích thước (tính bằng byte) của các khóa và bản mã của ML-KEM

	Khóa đóng gói	Khóa giải gói	Bản mã	Khóa bí mật được chia sẻ
ML-KEM-768	1184	2400	1088	32
ML-KEM-1024	1568	3168	1568	32

Phụ lục A

(tham khảo)

Tài liệu hỗ trợ và hướng dẫn triển khai ứng dụng cho ML-KEM

A.1. Các mức an toàn và tài liệu hỗ trợ

ML-KEM được kế thừa từ phiên bản vòng thứ ba của KEM CRYSTALS-KYBER [1], một bản đề trình trong quy trình tiêu chuẩn hóa mã hóa hậu lượng tử của NIST. Đặc tả trong tiêu chuẩn này tương thích với ML-KEM được quy định trong FIPS 203 [2]. Trong quy trình tiêu chuẩn hóa mã hóa hậu lượng tử, NIST đã định nghĩa năm mức an toàn được tóm tắt trong Bảng A.1-1.

Bảng A.1-1 – Năm mức an toàn

Mức an toàn	Loại tấn công tương ứng	Ví dụ
1	Tìm kiếm khóa trên mã khối với khóa 128 bit	AES-128
2	Tìm kiếm và chạm trên hàm băm 256 bit	SHA3-256
3	Tìm kiếm khóa trên mã khối với khóa 192 bit	AES-192
4	Tìm kiếm và chạm trên hàm băm 384 bit	SHA3-384
5	Tìm kiếm khóa trên mã khối với khóa 256 bit	AES-256

Trong FIPS 203, ML-KEM được định nghĩa cho các tham số ở các mức an toàn 1, 3 và 5. Các bộ tham số ML-KEM-768 và ML-KEM-1024 được định nghĩa trong tiêu chuẩn này được coi là thuộc các mức an toàn tương ứng là 3 và 5, như đã nêu trong [3]. Tài liệu [4] và [1] thảo luận về tính an toàn IND-CCA2 và tính chính xác của ML-KEM.

Các điểm khác biệt giữa bản đề trình vòng ba của CRYSTALS-KYBER và ML-KEM được liệt kê trong Phụ lục C của Tài liệu tham khảo [3]. Nguyên lý thiết kế của những điểm khác biệt này được thảo luận trong [5] và [6].

Độ an toàn của ML-KEM dựa trên độ khó giả định của việc giải bài toán MLWE [7], một dạng tổng quát hóa của bài toán học với lỗi (LWE) được Regev giới thiệu vào năm 2005 [8]. Độ khó của bài toán MLWE dựa trên độ khó giả định của một số bài toán tính toán nhất định trên lưới mô-đun [7]. Các kết quả nghiên cứu được công bố trong các Tài liệu tham khảo [9], [10], [11] và [12] cung cấp phân tích an toàn về độ khó của bài toán LWE, vốn là nền tảng cho độ an toàn của ML-KEM, các lược đồ và biến thể dựa trên LWE khác. Việc áp dụng một phiên bản cải tiến của tấn công sàng - đối ngẫu vào ML-KEM đã được nghiên cứu trong Tài liệu tham khảo [13]. Tuy nhiên, kết quả chính đã bị đặt dấu hỏi trong Tài liệu tham khảo [14]. Sau đó, độ phức tạp của tấn công sàng - đối ngẫu đã được nghiên cứu sâu hơn (ví dụ: trong các Tài liệu tham khảo [15] và [16]).

Bất kỳ việc nâng cấp nào từ mã hóa truyền thống lên cơ chế đóng gói khóa (KEM) này đều nên giữ lại mã hóa truyền thống và thêm KEM này như một lớp phòng thủ bổ sung, thay vì loại bỏ mã hóa truyền thống.

A.2. Lỗi giải gói khóa

Với điều kiện tất cả các đầu vào đều đúng cấu trúc, thủ tục thiết lập khóa của ML-KEM không bao giờ báo lỗi một cách tường minh. Tuy nhiên, có khả năng (dù cực kỳ khó xảy ra) quy trình này thất bại theo nghĩa là Alice (thông qua ML-KEM.Decaps) và Bob (thông qua ML-KEM.Encaps) đưa ra các đầu ra khác nhau, ngay cả khi cả hai bên đều hoạt động chính xác và không có sự can thiệp của kẻ tấn công. Trong trường hợp này, Alice và Bob rõ ràng đã không thành công trong việc đưa ra một khóa bí mật được chia sẻ. Sự kiện này được gọi là một lỗi giải gói khóa.

Xác suất lỗi giải gói khóa được định nghĩa là xác suất mà quy trình:

1. $(ek, dk) \leftarrow \text{ML-KEM.KeyGen}()$
2. $(c, K) \leftarrow \text{ML-KEM.Encaps}(ek)$
3. $K' \leftarrow \text{ML-KEM.Decaps}(c, dk)$

dẫn đến $K \neq K'$ (nghĩa là khóa được đóng gói khác với khóa được giải gói). Các ước tính về xác suất (hoặc tỷ lệ) lỗi giải gói khóa cho mỗi bộ tham số ML-KEM được đưa ra trong Bảng A.2-1 (xem [2]).

Bảng A.2-1 – Tỷ lệ lỗi giải gói khóa

Bộ tham số	Tỷ lệ lỗi giải gói khóa
ML-KEM-768	$2^{-164,8}$
ML-KEM-1024	$2^{-174,8}$

Tài liệu tham khảo

1. Bos, J.W., et al., *CRYSTALS-Kyber Algorithm Specifications and Supporting Documentation (Round 3)*. 2021, Technical report, CRYSTALS Team, January 2021. URL <https://pq-crystals.org>
2. FIPS203, *Module-Lattice-Based Key-Encapsulation Mechanism Standard*. 2024.
3. Fips, N., *203: Module-lattice-based key-encapsulation mechanism standard*. National Institute of Standards and Technology: Gaithersburg, MD, USA, 2024.
4. Bacelar Almeida, J., et al., *Formally verifying Kyber–Episode V: Machine-checked IND-CCA security and correctness of ML-KEM in EasyCrypt*. 2024.
5. *Federak Register Notice Department of Commerce, National Institute of Standards and Technology. Announcing Issuance of Federal Information Processing Standards (FIPS) FIPS 203, Module-Lattice-Based Key-Encapsulation Mechanism Standard, FIPS 204, Module-Lattice-Based Digital Signature Standard, and FIPS 205, Stateless Hash-Based Digital Signature Standard*.
6. *National Institute of Standards and Technology PQC Team, Presentation at the 5th NIST PQC standardization workshop*.
7. Langlois, A. and D. Stehlé, *Worst-case to average-case reductions for module lattices*. *Designs, Codes and Cryptography*, 2015. **75**(3): p. 565-599.
8. Regev, O., *On lattices, learning with errors, random linear codes, and cryptography*. *Journal of the ACM (JACM)*, 2009. **56**(6): p. 1-40.
9. Albrecht, M.R., et al. *Estimate all the {LWE, NTRU} schemes!* in *International Conference on Security and Cryptography for Networks*. 2018. Springer.
10. Albrecht, M.R., et al. *The general sieve kernel and new records in lattice reduction*. in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 2019. Springer.
11. Dachman-Soled, D., et al. *LWE with side information: attacks and concrete security estimation*. in *Annual international cryptology conference*. 2020. Springer.
12. Albrecht, M.R., et al. *Estimating quantum speedups for lattice sieves*. in *International Conference on the Theory and Application of Cryptology and Information Security*. 2020. Springer.
13. MATZOV, I., *Report on the security of lwe: improved dual lattice attack*, 2022. URL: <https://zenodo.org/record/6412487>.
14. Ducas, L. and L.N. Pulles. *Does the dual-sieve attack on learning with errors even work?* in *Annual International Cryptology Conference*. 2023. Springer.
15. Ducas, L. and L.N. Pulles, *Accurate score prediction for dual-sieve attacks*. *Cryptology ePrint Archive*, 2023.
16. Wiemers, A. and K. Bashiri, *On the Independence Heuristic in the Dual Attack*. *Cryptology ePrint Archive*, 2023.