FIPS 203 (borrador)

Publicación de estándares federales de procesamiento de información

Basado en módulo de celosía Encapsulación de claves Estándar del mecanismo

Categoría: Seguridad Informática Subcategoría: Criptografía

Laboratorio de Tecnología de la Información
Instituto Nacional de Estándares y Tecnología
Gaithersburg, MD 20899-8900

Esta publicación está disponible de forma gratuita en:
https://doi.org/10.6028/NIST.FIPS.203.ipd

Publicado el 24 de agosto de 2023

Departamento de Comercio de EE.
UU. Gina M. Raimondo, Secretaria

Instituto Nacional de Estándares y Tecnología Laurie E.
Locascio, directora del NIST y subsecretaria de Comercio de Estándares y Tecnología

Prefacio

La Serie de Publicaciones Federales de Estándares de Procesamiento de Información (FIPS) del Instituto Nacional de Estándares y Tecnología es la serie oficial de publicaciones relacionadas con estándares y pautas desarrolladas bajo 15 USC 278g-3 y emitida por el Secretario de Comercio bajo 40 USC 11331.

Los comentarios relacionados con esta publicación del Estándar Federal de Procesamiento de Información son bienvenidos y deben enviarse utilizando la información de contacto en la cláusula "Consultas y comentarios" de la sección de anuncios.

James A. St Pierre, director interino del
Laboratorio de Tecnología de la Información

Abstracto

Un mecanismo de encapsulación de claves (o KEM) es un conjunto de algoritmos que, bajo ciertas condiciones, pueden ser utilizados por dos partes para establecer una clave secreta compartida a través de un canal público. Una clave secreta compartida que se establece de forma segura mediante un KEM se puede utilizar con algoritmos criptográficos de clave simétrica para realizar tareas básicas en comunicaciones seguras, como cifrado y autenticación.

Este estándar especifica un mecanismo de encapsulación de claves llamado ML-KEM. La seguridad de ML-KEM está relacionada con la dificultad computacional del llamado Módulo de Aprendizaje con Errores. problema. En la actualidad, se cree que ML-KEM es seguro incluso contra adversarios que poseen una computadora cuántica.

Este estándar especifica tres conjuntos de parámetros para ML-KEM. En orden de mayor seguridad (y menor rendimiento), estos conjuntos de parámetros son ML-KEM-512, ML-KEM-768 y ML-KEM-1024.

Palabras clave: seguridad informática; criptografía; cifrado; Procesamiento de información federal Normas; criptografía basada en celosía; encapsulación de claves; poscuántico; criptografía de clave pública

Publicación 203 de normas federales de procesamiento de información

Publicado: 24 de agosto de 2023

anunciando el

Encapsulación de claves basada en módulos Estándar del mecanismo

Las publicaciones de estándares federales de procesamiento de información (FIPS) son emitidas por el Instituto Nacional de Estándares y Tecnología (NIST) según 15 USC 278g-3 y por el Secretario de Comercio según 40 USC 11331.

1. Nombre de la Norma. Estándar de mecanismo de encapsulación de claves basado en celosía de módulo (ML-KEM) (FIPS PUB 203).

2. Categoría de Norma. La seguridad informática. Subcategoría. Criptografía.

3. Explicación. Este estándar especifica un conjunto de algoritmos para aplicaciones que requieren una clave criptográfica secreta compartida por dos partes que solo pueden comunicarse a través de un canal público. Una clave criptográfica (o simplemente "clave") se representa en una computadora como una cadena de bits. Una clave secreta compartida la calculan conjuntamente dos partes (por ejemplo, la Parte A y la Parte B) utilizando un conjunto de reglas y parámetros. Bajo ciertas condiciones, estas reglas y parámetros garantizan que ambas partes producirán la misma clave y que esta clave compartida sea secreta para los adversarios. Una clave secreta compartida de este tipo se puede utilizar con algoritmos criptográficos de clave simétrica (especificados en otros estándares del NIST) para realizar tareas, como el cifrado y la autenticación de información digital.

   Si bien existen muchos métodos para establecer una clave secreta compartida, el método particular descrito en esta especificación es un mecanismo de encapsulación de claves (KEM). En un KEM, el cálculo de la clave secreta compartida comienza cuando la Parte A genera una clave de decapsulación y una clave de encapsulación. La parte A mantiene privada la clave de decapsulación y pone la clave de encapsulación a disposición de la parte B. La parte B luego usa la clave de encapsulación de la parte A para generar una copia de una clave secreta compartida junto con un texto cifrado asociado. Luego, la parte B envía el texto cifrado a la parte A a través del mismo canal. Finalmente, la Parte A utiliza el texto cifrado de la Parte B junto con la clave de decapsulación privada de la Parte A para calcular otra copia de la clave secreta compartida.

   La seguridad del KEM particular especificado aquí está relacionada con la dificultad computacional de resolver ciertos sistemas de ecuaciones lineales ruidosas, específicamente el llamado problema de aprendizaje de módulos con errores (MLWE). Actualmente, se cree que este método particular de establecer una clave secreta compartida es seguro incluso contra adversarios que posean una computadora cuántica. En el futuro, es posible que se especifiquen y aprueben KEM adicionales en publicaciones FIPS o en publicaciones especiales del NIST.

4. Autoridad Aprobatoria. Secretario de Comercio.

5. Agencia de Mantenimiento. Departamento de Comercio, Instituto Nacional de Estándares y Tecnología nología, Laboratorio de Tecnologías de la Información (ITL).

6. Aplicabilidad. Los Estándares Federales de Procesamiento de Información se aplican a los sistemas de información utilizados u operados por agencias federales o por un contratista de una agencia u otra organización en nombre de una agencia. No se aplican a los sistemas de seguridad nacional según se definen en 44 USC 3552.

   Este estándar debe implementarse siempre que se requiera el establecimiento de una clave secreta compartida para aplicaciones federales, incluido el uso de dicha clave con algoritmos criptográficos de clave simétrica, de acuerdo con la Oficina de Administración y Presupuesto aplicable y las políticas de la agencia. Las agencias federales también pueden utilizar métodos alternativos que el NIST haya indicado que son apropiados para este propósito.

   La adopción y el uso de esta norma están disponibles para organizaciones privadas y comerciales.

7. Implementaciones. Se puede implementar un mecanismo de encapsulación de claves en software, firmware, hardware o cualquier combinación de los mismos. Una implementación conforme puede reemplazar la secuencia dada de pasos en los algoritmos de nivel superior de ML-KEM (es decir, ML-KEM.KeyGen, ML-KEM.Encaps y ML-KEM.Decaps) con cualquier proceso equivalente. En otras palabras, se permiten diferentes procedimientos que produzcan la salida correcta para cada entrada. En particular, no se requiere que las implementaciones conformes utilicen las mismas subrutinas (de los algoritmos principales antes mencionados) que se utilizan en esta especificación.

   NIST desarrollará un programa de validación para probar la conformidad de las implementaciones con los algoritmos de este estándar. La información sobre los programas de validación está disponible en https: //csrc.nist.gov/projects/cmvp. Los valores de ejemplo para algoritmos criptográficos están disponibles en https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines/example-values.

8. Otras Funciones de Seguridad Aprobadas. Las implementaciones que cumplan con este estándar deberán emplear algoritmos criptográficos que hayan sido aprobados para proteger la información confidencial del gobierno federal . Los algoritmos y técnicas criptográficos aprobados incluyen aquellos que son:

   (a) Especificado en una publicación de Estándares Federales de Procesamiento de Información (FIPS),

   (b) Adoptado en una recomendación FIPS o NIST, o

   (c) Especificadas en la lista de funciones de seguridad aprobadas para FIPS 140-3.

9. Control de Exportaciones. Ciertos dispositivos criptográficos y los datos técnicos relacionados con ellos están sujetos a controles federales de exportación. Las exportaciones de módulos criptográficos que implementan este estándar y los datos técnicos relacionados con ellos deben cumplir con todas las leyes y regulaciones federales y estar autorizadas por la Oficina de Industria y Seguridad del Departamento de Comercio de EE. UU. La información sobre las regulaciones de exportación está disponible en https://www.bis.doc.gov.

10. Patentes. NIST ha celebrado dos acuerdos de licencia de patentes para facilitar la adopción de la selección anunciada por el NIST del algoritmo PQC de cifrado de clave pública CRYSTALS-KYBER. El NIST y las partes otorgantes de licencias comparten el deseo, en aras del interés público, de que las patentes con licencia estén disponibles gratuitamente para que las practique cualquier implementador del algoritmo ML-KEM publicado por el NIST. ML-KEM es el nombre que recibe el algoritmo en este estándar derivado de CRYSTALS- KYBER. Para obtener un resumen y extractos de la licencia, consulte https://csrc.nist.gov/csrc/m edia/Projects/post-quantum-cryptography/documents/selected-algos-2022/nist-pqc-license-summer- and-excerpts.pdf. La implementación del algoritmo especificado en el estándar puede estar cubierta por patentes estadounidenses y extranjeras de las que el NIST no tiene conocimiento.

11. Calendario de implementación. Esta norma entra en vigor inmediatamente después de la publicación final. ción.

12. Especificaciones. Estándares federales de procesamiento de información (FIPS) 203, basado en celosía de módulos Estándar del mecanismo de encapsulación de claves (adjunto).

13. Cualificaciones. En las aplicaciones, las garantías de seguridad de un KEM sólo son válidas bajo ciertas condiciones. condiciones (consulte NIST SP 800-227 [1]). Una de esas condiciones es el secreto de varios valores, incluida la aleatoriedad utilizada por las dos partes, la clave de decapsulación y el secreto compartido clave en sí. Por lo tanto, los usuarios deberán protegerse contra la divulgación de estos valores.

Si bien la intención de esta norma es especificar los requisitos generales para implementar Algoritmos ML-KEM , la conformidad con este estándar no garantiza que una implementación particular sea segura. Es responsabilidad del implementador garantizar que cualquier módulo que implementa una capacidad de establecimiento clave está diseñada y construida de manera segura.

De manera similar, el uso de un producto que contenga una implementación que cumpla con este estándar no garantiza la seguridad del sistema general en el que se utiliza el producto. La autoridad responsable de cada agencia o departamento deberá garantizar que una implementación general proporcione un nivel aceptable de seguridad.

El NIST seguirá siguiendo los avances en el análisis del algoritmo ML-KEM. Al igual que con sus otros estándares de algoritmos criptográficos, el NIST reevaluará formalmente este estándar cada cinco años.

Tanto este estándar como las posibles amenazas que reducen la seguridad proporcionada mediante el uso de este estándar serán revisados por el NIST según corresponda, teniendo en cuenta los análisis y la tecnología recientemente disponibles. Además, el conocimiento de cualquier avance tecnológico o cualquier debilidad matemática del algoritmo hará que el NIST reevalúe este estándar y proporcione las revisiones necesarias.

14. Procedimiento de Renuncia. La Ley Federal de Gestión de Seguridad de la Información (FISMA) no no permitir exenciones a los Estándares Federales de Procesamiento de Información (FIPS) que se realicen obligatorio por el Secretario de Comercio.

15. Dónde obtener copias de la norma. Esta publicación está disponible accediendo https://csrc.nist.gov/publications. Otras publicaciones sobre seguridad informática están disponibles en mismo sitio web.

16. Cómo citar esta publicación. NIST ha asignado NIST FIPS 203 ipd como publicación identificador para este FIPS, según la sintaxis de identificador de publicaciones de la serie técnica del NIST. NIST recomienda que se cite de la siguiente manera:

> Instituto Nacional de Estándares y Tecnología (2023) Estándar de mecanismo de encapsulación de clave basado en módulo . (Departamento de Comercio, Washington, DC), Publicación de estándares federales de procesamiento de información (FIPS) NIST FIPS 203 ipd. https://doi.org/10.6028/NIST.FIPS.203.ipd

17. Consultas y Comentarios. Las consultas y comentarios sobre este FIPS pueden enviarse a fps-203-comments@nist.gov.

Convocatoria de reclamaciones de patentes

Esta revisión pública incluye una convocatoria de información sobre reivindicaciones de patentes esenciales (reivindicaciones cuyo uso sería necesario para cumplir con la orientación o los requisitos de este borrador de publicación del Laboratorio de Tecnología de la Información (ITL)). Dichas orientaciones y/o requisitos pueden indicarse directamente en esta Publicación ITL o mediante referencia a otra publicación. Esta convocatoria también incluye la divulgación, cuando se conozca, de la existencia de solicitudes de patentes estadounidenses o extranjeras pendientes relacionadas con este borrador de publicación ITL y de cualquier patente estadounidense o extranjera relevante y vigente.

ITL podrá exigir al titular de la patente, o a una parte autorizada para dar garantías en su nombre, en forma escrita o electrónica, ya sea:

a) garantía en forma de descargo de responsabilidad general en el sentido de que dicha parte no posee y actualmente no tiene la intención de poseer ninguna reivindicación de patente esencial; o

b) garantía de que una licencia para dichas reivindicaciones de patente esenciales estará disponible para los solicitantes que deseen utilizar la licencia con el fin de cumplir con la guía o requisitos en este borrador de publicación del DIT:

(i) bajo términos y condiciones razonables que estén demostrablemente libres de cualquier trato injusto discriminación; o

(ii) sin compensación y bajo términos y condiciones razonables que estén demostrablemente libres de cualquier discriminación injusta.

Dicha garantía deberá indicar que el titular de la patente (o un tercero autorizado para dar garantías en su nombre) incluirá en cualquier documento de transferencia de propiedad de patentes sujetas a la garantía, disposiciones suficientes para garantizar que los compromisos de la garantía sean vinculantes para el cesionario. y que el cesionario incluirá igualmente disposiciones apropiadas en caso de futuras transferencias con el objetivo de vincular a cada sucesor en interés.

La garantía también indicará que está destinada a ser vinculante para los sucesores en interés independientemente de si dichas disposiciones están incluidas en los documentos de transferencia pertinentes.

Dichas declaraciones deben dirigirse a: fps-203-comments@nist.gov.

Publicación 203 de normas federales de procesamiento de información

Especificación para el

Encapsulación de claves basada en módulos Estándar del mecanismo

Tabla de contenido

Machine Translated by Google

Lista de tablas

Lista de Figuras

Lista de algoritmos

1. Introducción

1.1 Propósito y Alcance

Este estándar especifica el mecanismo de encapsulación de claves basado en módulo de celosía, o ML-KEM. Un mecanismo de encapsulación de claves (o KEM) es un conjunto de algoritmos que se pueden utilizar para establecer una clave secreta compartida entre dos partes que se comunican a través de un canal público. Un KEM es un tipo particular de esquema de establecimiento de claves. Los esquemas actuales de establecimiento de claves aprobados por el NIST se especifican en NIST SP-800-56A, Recomendación para esquemas de establecimiento de claves por pares. Uso de criptografía basada en logaritmos discretos [2] y NIST SP-800-56B, Recomendación para Esquemas de establecimiento de claves por pares que utilizan criptografía de factorización de enteros [3].

Es bien sabido que los esquemas de establecimiento de claves especificados en NIST SP-800-56A y NIST SP-800-56B son vulnerables a ataques que utilizan computadoras cuánticas suficientemente capaces. ML-KEM es una alternativa aprobada que actualmente se cree que es segura incluso contra adversarios en posesión de una computadora cuántica. ML-KEM se deriva de la versión de tercera ronda de CRYSTALS-KYBER KEM [4], una presentación en el proyecto de estandarización de criptografía poscuántica del NIST . Para conocer las diferencias entre ML-KEM y CRYSTALS-KYBER, consulte la Sección 1.3.

Este estándar especifica los algoritmos y conjuntos de parámetros del esquema ML-KEM . Su objetivo es proporcionar información suficiente para implementar ML-KEM de una manera que pueda pasar la validación (consulte https://csrc.nist.gov/projects/cryptographic-module-validation-program). Para conocer las definiciones y propiedades generales de los KEM, incluidos los requisitos para el uso seguro de los KEM en aplicaciones, consulte NIST SP 800-227 [1].

Este estándar especifica tres conjuntos de parámetros para ML-KEM. Estos conjuntos de parámetros ofrecen diferentes compensaciones entre la solidez de la seguridad y el rendimiento. Los tres conjuntos de parámetros de ML-KEM están aprobados para proteger sistemas de comunicación sensibles y no clasificados del gobierno federal de EE. UU.

1.2 Contexto

En los últimos años, ha habido un progreso constante hacia la construcción de computadoras cuánticas. Si se construyen computadoras cuánticas a gran escala, la seguridad de muchos criptosistemas de clave pública de uso común estará en riesgo. Esto incluiría esquemas de establecimiento de claves y esquemas de firma digital que se basan en la factorización de números enteros y logaritmos discretos (tanto en campos finitos como en curvas elípticas). Como resultado, en 2016, el Instituto Nacional de Estándares y Tecnología (NIST) inició un proceso público para seleccionar algoritmos criptográficos de clave pública resistentes a los cuánticos para su estandarización. Se envió un total de 82 algoritmos candidatos al NIST para su consideración para su estandarización.

Después de tres rondas de evaluación y análisis, el NIST seleccionó los primeros cuatro algoritmos para estandarizarlos como resultado del proceso de estandarización de la criptografía poscuántica (PQC). Estos algoritmos están destinados a proteger la información confidencial del gobierno de los EE. UU. durante mucho tiempo en el futuro previsible, incluso después de la llegada de las computadoras cuánticas. Este estándar especifica una variante del algoritmo seleccionado CRYSTALS-KYBER, un mecanismo de encapsulación de claves basado en celosía (KEM) [4]. En este estándar, el KEM especificado aquí se denominará ML-KEM, ya que se basa en el llamado supuesto de Aprendizaje de Módulos con Errores.

1.3 Diferencias con la presentación de CRYSTALS-KYBER

A continuación se muestra una lista de todas las diferencias de esquema entre CRYSTALS-KYBER (como se describe en [4]) y el esquema ML-KEM especificado en este documento. La lista consta únicamente de aquellas diferencias que resultan en un comportamiento diferente de entrada y salida de los algoritmos principales (es decir, KeyGen, Encaps, Decaps) de CRYSTALS-KYBER y ML-KEM. Recuerde que una implementación conforme solo necesita coincidir con el comportamiento de entrada-salida de estos tres algoritmos (consulte "Implementaciones" más arriba y la Sección 3.3 a continuación). En consecuencia, la siguiente lista no incluye ninguna de las numerosas diferencias en cómo los algoritmos principales realmente producen salidas a partir de entradas (por ejemplo, a través de diferentes pasos computacionales o diferentes subrutinas). La siguiente lista tampoco incluye ninguna diferencia en la presentación entre esta norma y [4].

- En la especificación de la tercera ronda [4], la clave secreta compartida se trató como un valor de longitud variable cuya longitud depende de cómo se usaría esta clave en la aplicación relevante. En esta especificación, la longitud de la clave secreta compartida se fija en 256 bits. En esta especificación, esta clave se puede utilizar directamente en aplicaciones como clave simétrica; alternativamente, se pueden derivar claves simétricas a partir de esta clave, como se especifica en la Sección 3.3.

- Los algoritmos ML-KEM.Encaps y ML-KEM.Decaps en esta especificación utilizan una variante diferente de la transformada Fujisaki-Okamoto (ver [5, 6]) que la especificación de tercera ronda [4]. Específicamente, ML-KEM.Encaps ya no incluye un hash del texto cifrado en la derivación del secreto compartido y ML-KEM.Decaps se ha ajustado para que coincida con este cambio.

- En la especificación de tercera ronda [4], la aleatoriedad inicial m en ML-KEM.Encaps El algoritmo fue codificado por primera vez antes de ser utilizado. Específicamente, entre las líneas 1 y 2 del Algoritmo 16, hubo un paso adicional que realizó la operación $m \leftarrow H(m)$. El propósito de este paso era proteger contra el uso de procesos de generación de aleatoriedad defectuosos. Como este estándar requiere el uso de generación de aleatoriedad aprobada por NIST, este paso es innecesario y no se realiza en ML-KEM.

- Esta especificación incluye pasos explícitos de validación de entradas que no formaban parte de la especificación de la tercera ronda [4]. Por ejemplo, ML-KEM.Encaps requiere que la matriz de bytes que contiene la clave de encapsulación se decodifique correctamente en una matriz de enteros módulo q sin ninguna reducción modular.

2. Glosario de términos, acrónimos y símbolos matemáticos

2.1 Términos y definiciones

aprobado	Aprobado por FIPS y/o recomendado por NIST. Un algoritmo o técnica que 1) se especifica en una recomendación FIPS o NIST, 2) se adopta en una recomendación FIPS o NIST, o 3) se especifica en una lista de funciones de seguridad aprobadas por NIST.
decapsulación	El proceso de aplicación del algoritmo Decaps de un KEM. Este algoritmo acepta un texto cifrado KEM y la clave de decapsulación como entrada y produce una clave secreta compartida como salida.
clave de decapsulación	Clave criptográfica producida por un KEM durante la generación de claves y utilizada durante el proceso de decapsulación. La clave de decapsulación debe mantenerse privada y debe destruirse cuando ya no sea necesaria.
clave de descifrado	Una clave criptográfica que se utiliza con un PKE para descifrar textos cifrados en textos sin formato. La clave de descifrado debe mantenerse privada y debe destruirse cuando ya no sea necesaria.
destruir	Una acción aplicada a una clave u otro dato secreto. Una vez destruido un dato secreto, no se puede recuperar ninguna información sobre su valor.
encapsulación	El proceso de aplicación del algoritmo Encaps de un KEM. Este algoritmo acepta la aleatoriedad privada y la clave de encapsulación como entrada y produce una clave secreta compartida y un texto cifrado asociado como salida.
clave de encapsulación	Clave criptográfica producida por un KEM durante la generación de claves y utilizada durante el proceso de encapsulación. La clave de encapsulación se puede hacer pública.
Clave de encriptación	Una clave criptográfica que se utiliza con un PKE para cifrar textos sin formato en textos cifrados. La clave de cifrado se puede hacer pública.
proceso equivalente	Dos procesos son equivalentes si se produce la misma salida cuando se ingresan los mismos valores a cada proceso (ya sea como parámetros de entrada, como valores disponibles durante el proceso, o ambos).
función hash	Una función en cadenas de bits en la que la longitud de la salida es fija. Las funciones hash aprobadas relevantes para este estándar se especifican en FIPS 202 [7].
Texto cifrado KEM	Una cadena de bits que se produce mediante encapsulación y se utiliza como entrada para la decapsulación.
llave	Una cadena de bits que se utiliza junto con un algoritmo criptográfico. Los ejemplos aplicables a este estándar incluyen: las claves de encapsulación y desencapsulación (de un KEM), la clave secreta compartida (producida por un KEM) y las claves de cifrado y descifrado (de una PKE).

mecanismo de encapsulación de claves (KEM)	Un conjunto de tres algoritmos criptográficos (KeyGen, Encaps y Decaps) que pueden utilizar dos partes para establecer una clave secreta compartida a través de un canal público.
Par de claves	Un conjunto de dos claves con la propiedad de que una clave puede hacerse pública mientras que la otra debe mantenerse privada. En este estándar, esto podría referirse al par de claves (clave de encapsulación, clave de desencapsulación) de un KEM o al par de claves (clave de cifrado, clave de descifrado) de un PKE.
fiesta	Un individuo (persona), organización, dispositivo o proceso. En esta especificación , hay dos partes (Parte A y Parte B, o Alice y Bob), y realizan conjuntamente el proceso de establecimiento de claves utilizando un KEM.
pseudoaleatorio	Se dice que un proceso (o datos producidos por un proceso) es pseudoaleatorio cuando el resultado es determinista pero también parece aleatorio siempre que la acción interna del proceso esté oculta a la observación. Para fines criptográficos, "efectivamente aleatorio" significa "computacionalmente indistinguible de aleatorio dentro de los límites de la seguridad prevista ".
canal publico	Un canal de comunicación entre dos partes; dicho canal puede ser observado y posiblemente también dañado por terceros.
esquema de cifrado de clave pública (PKE)	Un conjunto de tres algoritmos criptográficos (KeyGen, Encrypt y Decrypt) que pueden utilizar dos partes para enviar datos secretos a través de un canal público. También conocido como esquema de cifrado asimétrico.
clave secreta compartida	El resultado final de un proceso de establecimiento de clave KEM. Es una clave criptográfica que se puede utilizar para criptografía de clave simétrica. Debe mantenerse en privado y debe destruirse cuando ya no sea necesario.
categoría de seguridad	Un número asociado con la fuerza de seguridad de un algoritmo criptográfico poscuántico según lo especificado por NIST (consulte el Apéndice A, Tabla 4).
Fuerza de seguridad	Un número asociado con la cantidad de trabajo que se requiere para romper un algoritmo o sistema criptográfico.
deberá	Se utiliza para indicar un requisito de esta norma.
debería	Se utiliza para indicar una recomendación fuerte pero no un requisito de esta norma. Ignorar la recomendación podría conducir a resultados indeseables.

2.2 Acrónimos

AES	Estándar de cifrado avanzado
CDB	Distribución Binomial Centrada
FIPS	Estándar federal de procesamiento de información

KEM	Mecanismo de encapsulación de claves
LWE	Aprender con errores
MLWE	Módulo de aprendizaje con errores
NIST	Instituto Nacional de Estándares y Tecnología
nistir	Informe interinstitucional o interno del NIST
NTT	Transformada teórica de números
PKE	Cifrado de clave pública
PQC	Criptografía poscuántica
PRF	Función pseudoaleatoria
RBG	Generador de bits aleatorios
sha	Algoritmo hash seguro
AGITAR	Algoritmo hash seguro KECCAK
SP	Publicación especial
XOF	Función de salida extensible

2.3 Símbolos matemáticos

S	Si S es un conjunto, esto denota el conjunto de tuplas (o matrices) de longitud finita de elementos del conjunto S, incluida la tupla vacía (o matriz vacía).
S^k	Si S es un conjunto, esto denota el conjunto de k-tuplas (o matrices de longitud-k) de elementos del conjunto S.
BitRev7(r)	Inversión de bits de un entero r de siete bits . Específicamente, si $r = r_0 + 2r_1 + 4r_2 + \cdots + 64r_6$ con r_i {0,1}, entonces BitRev7(r) $= r_6 + 2r_5 + 4r_4 + \cdots + 64r_0$.
$\hat{F}$	El elemento de T_q que es igual a la representación NTT de un polinomio f R_q (ver Sección 4.3).
q	El conjunto de los números racionales.
zm	El anillo de números enteros módulo m, es decir, el conjunto {0,1,...,m−1} equipado con las operaciones de suma y multiplicación módulo m.
z	El conjunto de los números enteros.
Vermont , EN	La transpuesta de una fila o columna v; además, la transpuesta de una matriz A.
fj	El coeficiente de X^j de un polinomio $f = f_0 + f_1X + \cdots + f_{255}X^{255}$ R_q.
r mod m	El entero único r′ en {0,1,...,m−1} tal que m divide r −r′ .

r mod ± m	Para m par (respectivamente, impar), esto denota el entero único r′ tal que −m/2 < r′ ≤ m/ 2 (respectivamente, −(m−1)/2 ≤ r′ ≤ (m−1)/ 2) y m divide r −r′ .
|B|	Si B es un número, esto denota el valor absoluto de B. Si B es una matriz, esto denota su longitud.
	El techo de x, es decir, el número entero más pequeño mayor o igual a x.
x x	El redondeo de x al número entero más cercano; si x = y+1/2 para algún y Z, entonces x = y+1.
x	El piso de x, es decir, el mayor entero menor o igual a x.
B	El conjunto {0,1,...,255} de enteros de 8 bits sin signo (bytes).
A B	La concatenación de dos matrices o cadenas de bits A y B.
Bi]	La entrada en el índice i en la matriz B. Todas las matrices tienen índices que comienzan en cero.
B[k: m]	El subarreglo (B[k],B[k +1],...,B[m−1]) del arreglo B.
norte	Denota el número entero 256 en todo este documento.
q	Denota el número entero primo 3329 = 28 · 13+1 en todo este documento.
rq	El anillo Zq[X]/(Xn + 1) que consta de polinomios de la forma f = f0 + f1X + ··· + f255X^{255} donde fj Zq para todo j, equipado con módulo de suma y multiplicación Xn +1.
s ← x	En pseudocódigo, esta notación significa que a la variable s se le asigna el valor de la expresión x.
$ s ←− Bℓ	En pseudocódigo, esta notación significa que a la variable s se le asigna el valor de una matriz de ℓ bytes aleatorios. Los bytes deben generarse mediante aleatoriedad a partir de un RBG aprobado (consulte la Sección 3.3).
tq	La imagen de Rq bajo la transformada de teoría de números. Sus elementos se denominan "representaciones NTT" de polinomios en Rq (ver la Sección 4.3).

2.4 Interpretación del pseudocódigo

Esta sección describe las convenciones del pseudocódigo utilizado para describir los algoritmos de este estándar. Se entiende que todos los algoritmos tienen acceso a dos constantes enteras globales: n = 256 y q = 3329. También hay cinco variables enteras globales: k, η1, η2, du y dv. Todas las demás variables son locales. Las cinco variables globales se establecen en valores particulares cuando se selecciona un conjunto de parámetros (consulte la Sección 7).

Cuando los algoritmos de esta especificación invocan otros algoritmos como subrutinas, todos los argumentos (entradas) se pasan por valor. En otras palabras, se crea una copia de las entradas y con la copia se invoca la subrutina. No existe el "pasar por referencia".

Tipos de datos. Para variables que representan la entrada o salida de un algoritmo, el tipo de datos (p. ej., bit, byte, matriz de bits) se describirán explícitamente al comienzo del algoritmo. Para la mayoría de las variables locales del pseudocódigo, el tipo de datos se deduce fácilmente del contexto. Para todas las demás variables, el tipo de datos se declarará en un comentario. En un algoritmo único, el tipo de datos de una variable se determina la primera vez que se utiliza la variable y no se cambiará. Los nombres de variables pueden reutilizarse y se reutilizarán en diferentes algoritmos, incluso con diferentes tipos de datos.

Además de los tipos de datos atómicos estándar (p. ej., bits, bytes) y estructuras de datos (p. ej., matrices), también se utilizarán enteros módulo m (es decir, elementos de $\mathbb{Z}_m$) como tipo de datos abstractos. Está implícito que la reducción del módulo m tiene lugar siempre que se realiza una asignación a una variable en $\mathbb{Z}_m$. Por ejemplo, para $z \in \mathbb{Z}_m$ y cualquier número entero x, y, la declaración

$$z \leftarrow x+y \tag{2.1}$$

significa que a z se le asigna el valor x+y mod m. El pseudocódigo es independiente con respecto a cómo se representa un módulo entero m en implementaciones reales o cómo se calcula la reducción modular.

Sintaxis de bucle. El pseudocódigo utilizará los bucles " while " y " for ". La sintaxis del "mientras" se explica por sí misma. En el caso de los bucles "for", la sintaxis será del estilo del lenguaje de programación C. En el Algoritmo 1 se dan dos ejemplos sencillos .

Algoritmo 1 para ejemplo

Realiza dos bucles "for" simples.

```
para (i ← 0; i < 10; i++)
    A[i] ← i                      ▷ A es una matriz de números enteros de longitud 10
final para                        ▷ A ahora tiene el valor (0,1,2,3,4,5,6,7,8,9)
j ← 0
para (k ← 256; k > 1; k ← k/2)
    B[j] ← k                      ▷ B es una matriz de números enteros de longitud 8
    j ← j+1
final para                        ▷ B ahora tiene el valor (256,128,64,32,16,8,4,2)
```

Aritmética con matrices de números enteros. Este estándar hace un uso extensivo de matrices de números enteros módulo m (es decir, elementos de $\mathbb{Z}^{\ell}_m$). En un caso típico, los valores relevantes son m = q y ℓ = n = 256. Aritmética con matrices en $\mathbb{Z}^{\ell}_m$ se realizará de la siguiente manera. Sean $a \in \mathbb{Z}_m$ y $X, Y \in \mathbb{Z}^{\ell}_m$. Las declaraciones

$$Z \leftarrow a \cdot X$$

$$W \leftarrow X + Y$$

dará como resultado dos matrices $Z, W \in \mathbb{Z}^{\ell}_m$, con la propiedad de que $Z[i] = a \cdot X[i]$ y $W[i] = X[i]+Y[i]$ para todos yo. Multiplicación de matrices en $\mathbb{Z}^{\ell}_m$ sólo tendrá sentido cuando m = q y ℓ = n = 256, en cuyo caso corresponde a la multiplicación en un anillo particular. Esta operación se describirá en (2.2) a continuación.

Representaciones de objetos algebraicos. Una operación esencial en ML-KEM es la numeración.

transformada teórica (NTT), que asigna un polinomio f en un determinado anillo R_q a su "representación NTT" $\hat{f}$ en un anillo diferente T_q. Los anillos R_q y T_q y el NTT se analizan en detalle en la Sección 4.3. Este estándar representará elementos de R_q y elementos de T_q en pseudocódigo utilizando matrices de números enteros módulo q, de la siguiente manera.

Un elemento f de R_q es un polinomio de la forma

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

y será representado en pseudocódigo por la matriz

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

cuyas entradas contienen los coeficientes de f . Abusando un poco de la notación, esta matriz también se denotará por f . La entrada i-ésima de la matriz f contendrá así el coeficiente i-ésimo del polinomio f (es decir, $f[i] = f_i$).

Un elemento (a veces llamado "representación NTT") $\hat{g}$ de T_q es una tupla de 128 polinomios, cada uno de grado como máximo uno. Específicamente,

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

Un objeto algebraico de este tipo estará representado en pseudocódigo por la matriz

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

Abusando un poco de la notación, esta matriz también se denotará por $\hat{g}$. En este caso, la correspondencia entre las entradas de la matriz y los coeficientes es $\hat{g}[2i] = \hat{g}_{i,0}$ y $\hat{g}[2i+1] = \hat{g}_{i,1}$ para $i \in \{0,1,\dots,127\}$.

Conversión entre un polinomio $f \in R_q$ y su representación NTT $\hat{f} \in T_q$ se realizará mediante los algoritmos NTT (Algoritmo 8) y NTT−1 (Algoritmo 9). Estos algoritmos actúan sobre matrices de coeficientes, como se describió anteriormente, y satisfacen $\hat{f} = \mathrm{NTT}(f)$ y $f = \mathrm{NTT}^{-1}(\hat{f})$.

Aritmética con polinomios y representaciones NTT. Las operaciones algebraicas de suma y multiplicación escalar en R_q y T_q se realizan por coordenadas. Por ejemplo, si $a \in \mathbb{Z}_q$ y $f \in R_q$, el coeficiente i-ésimo del polinomio $a \cdot f \in R_q$ es igual a $a \cdot f_i \bmod q$. En pseudocódigo, los elementos tanto de R_q como de T_q se representan mediante matrices de coeficientes (es decir, elementos de $\mathbb{Z}_q^{256}$), como se describió anteriormente. Las operaciones algebraicas de suma y multiplicación escalar se realizan así mediante la suma y multiplicación escalar de las matrices de coeficientes correspondientes. Por ejemplo, la suma de dos representaciones NTT en pseudocódigo se realiza mediante una declaración de la forma $\hat{h} \leftarrow \hat{f} + \hat{g}$, donde $\hat{h}, \hat{f}, \hat{g} \in \mathbb{Z}_q^{256}$ son matrices de coeficientes.

Las operaciones algebraicas de multiplicación en R_q y T_q se tratan de la siguiente manera. Por motivos de eficiencia no se utilizará la multiplicación en R_q . El significado algebraico de la multiplicación en T_q se analiza en la Sección 4.3.1. En pseudocódigo, será realizado por el algoritmo MultiplyNTTs (Algoritmo 10). Específicamente, si $\hat{f}, \hat{g} \in \mathbb{Z}_q^{256}$ son un par de matrices (cada una representa el NTT de algún polinomio), entonces

$$\hat{h} \leftarrow \hat{f} \times_{T_q} \hat{g} \quad \text{medio} \quad \hat{h} \leftarrow \text{Multiplicar NTT}(\hat{f}, \hat{g}). \tag{2.2}$$

El resultado es una matriz $\hat{h} \in \mathbb{Z}_q^{256}$.

Matrices y vectores. Además de matrices de números enteros módulo q, el pseudocódigo también utilizará matrices cuyas entradas sean en sí mismas elementos de $\mathbb{Z}_q^{256}$. Por ejemplo, un elemento $v \in (\mathbb{Z}_q^{256})^3$ será una matriz de longitud tres cuyas entradas v[0], v[1] y v[2] son en sí mismas elementos de $\mathbb{Z}_q^{256}$ (es decir, matrices). Se puede pensar que cada una de estas entradas representa un polinomio en R_q, de modo que v en sí mismo representa un elemento del módulo R_q^3.

Cuando se utilizan matrices para representar matrices y vectores cuyas entradas son elementos de R_q, se indicarán con letras en negrita (por ejemplo, **v** para vectores y **A** para matrices). Cuando se utilizan matrices para representar matrices y vectores cuyas entradas son elementos de T_q, se denotarán con un "sombrero" (p. ej., $\hat{v}$ y $\hat{A}$). A menos que se realice una operación de transposición explícita, se entiende que los vectores son vectores de columna. Entonces se pueden ver los vectores como el caso especial de matrices con una sola columna.

La conversión entre matrices sobre R_q y matrices sobre T_q se realizará por coordenadas. Específicamente , si $A \in (\mathbb{Z}_q^{256})^{k\times\ell}$, entonces el enunciado

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

dará como resultado $\hat{A} \in (\mathbb{Z}_q^{256})^{k\times\ell}$ con $\hat{A}[i, j] = \text{NTT}(A[i, j])$ para todo i, j. Esto implica ejecutar NTT un total de $k \cdot \ell$ veces. Tenga en cuenta que el caso de los vectores corresponde a $\ell = 1$.

Aritmética con matrices y vectores. A continuación se describe cómo realizar aritmética con matrices sin dejar de ver los vectores como un caso especial de matrices.

La suma y la multiplicación escalar se realizan por coordenadas. Suma de matrices sobre R_q y entonces T_q es sencillo. En el caso de T_q, la multiplicación escalar se realiza mediante (2.2). Por ejemplo, si $\hat{f} \in \mathbb{Z}_q^{256}$ entonces y $\hat{u}, \hat{v} \in (\mathbb{Z}_q^{256})^k$,

$$\hat{w} \leftarrow \hat{f} \cdot \hat{u}$$
$$\hat{z} \leftarrow \hat{u} + \hat{v}$$

resultará en $\hat{w}, \hat{z} \in (\mathbb{Z}_q^{256})^k$ que satisface $\hat{w}[i] = \hat{f} \times_{T_q} \hat{u}[i]$ y $\hat{z}[i] = \hat{u}[i] + \hat{v}[i]$ para todo i. Tenga en cuenta que la multiplicación y suma de entradas individuales aquí se realiza utilizando la aritmética apropiada para matrices de coeficientes de elementos de T_q.

También será necesario multiplicar matrices con entradas en T_q. Esto se hace usando la multiplicación de matrices estándar, siendo la multiplicación del caso base (es decir, la multiplicación de entradas individuales) la multiplicación en T_q. Si $\hat{A}$ y $\hat{B}$ son dos matrices con entradas en T_q, su producto matricial se denotará $\hat{A} \circ \hat{B}$. A continuación se proporcionan algunos ejemplos de declaraciones en pseudocódigo que implican la multiplicación de matrices . En estos ejemplos, $\hat{A}$ es una matriz $k \times k$, mientras que $\hat{u}$ y $\hat{v}$ son vectores de longitud k. Todo

Tres de estos objetos están representados en pseudocódigo mediante matrices: una matriz $k \times k$ para $\hat{A}$ y longitud-k matrices para $\hat{u}$ y $\hat{v}$. Las entradas de $\hat{A}$, $\hat{u}$ y $\hat{v}$ son elementos de $\mathbb{Z}_q^{256}$. Las dos primeras declaraciones de pseudocódigo que aparecen a continuación producen un nuevo vector de longitud k cuyas entradas se especifican en el lado derecho . La tercera declaración de pseudocódigo calcula un producto escalar; por lo tanto, el resultado está en la base z del anillo $\mathbb{Z}_q^{256}$ (es decir, T_q) y está representado por un elemento .

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

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

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

La multiplicación $\times_{T_q}$ de las entradas individuales anteriores se realiza utilizando MultiplyNTT, como se describe en (2.2) arriba.

Aplicar algoritmos a arrays. Las convenciones de la aritmética de coordenadas descritas anteriormente también se extenderán a los algoritmos que actúan sobre (y/o producen) un tipo de datos atómicos. Cuando el pseudocódigo invoca dicho algoritmo en una entrada de matriz, se da a entender que el algoritmo se invoca repetidamente para cada entrada de la matriz. Por ejemplo, la función $\mathsf{Compress}_d : \mathbb{Z}_q \to \mathbb{Z}_{2^d}$ definido en la Sección 4 se puede invocar en una entrada de matriz $F \in \mathbb{Z}_q^{256}$ con la declaración

$$K \leftarrow \text{Comprimido}(F). \tag{2.3}$$

El resultado será que $K \in \mathbb{Z}_{2^d}^{256}$ y $K[i] = \mathsf{Compress}_d(F[i])$ para cada i. Este cálculo implica ejecutar el algoritmo Compress 256 veces.

3. Descripción general del esquema ML-KEM

Esta sección ofrece una descripción general de alto nivel del esquema ML-KEM.

3.1 Mecanismos clave de encapsulación

La siguiente es una descripción breve e informal de los mecanismos de encapsulación de claves (o KEM). Para obtener más detalles, consulte NIST SP 800-227 [1].

Un mecanismo de encapsulación de claves (o KEM) es un conjunto de algoritmos que pueden usarse, bajo ciertas condiciones, para establecer una clave secreta compartida entre dos partes que se comunican. Esta clave secreta compartida se puede utilizar para criptografía de clave simétrica.

Un KEM consta de tres algoritmos y una colección de conjuntos de parámetros. Los tres algoritmos son:

- un algoritmo de generación de claves denominado KeyGen;
- un algoritmo de "encapsulación" denominado Encaps;
- un algoritmo de "decapsulación" indicado por Decaps.

La colección de conjuntos de parámetros se utiliza para seleccionar un equilibrio entre seguridad y eficiencia. Cada parámetro establecido en la colección es una lista de valores numéricos específicos, uno para cada parámetro requerido por los algoritmos anteriores.

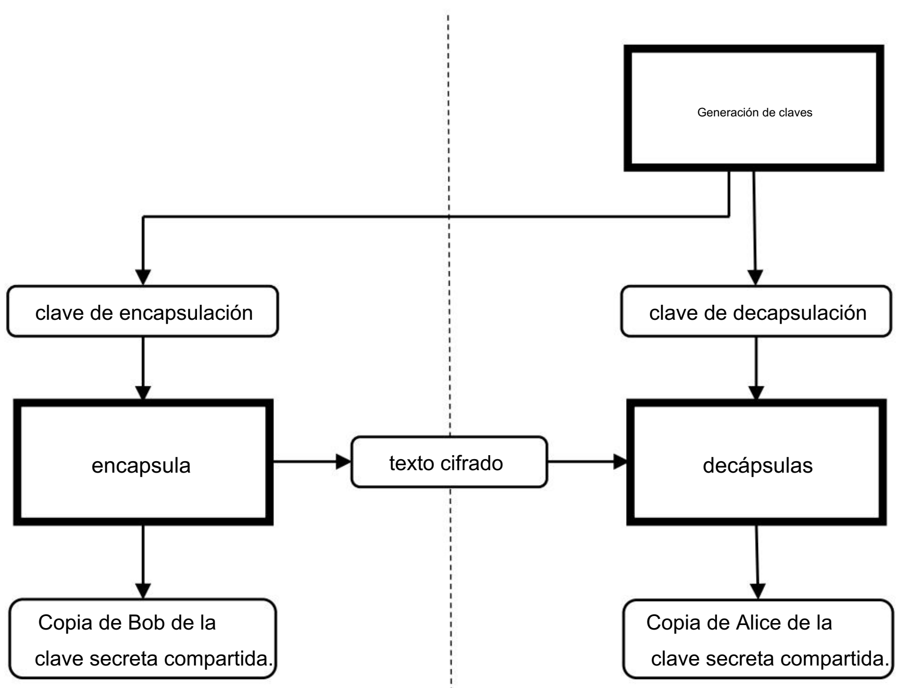

Figura 1. Una vista simple del establecimiento de claves utilizando un KEM

Se puede utilizar un KEM para establecer una clave secreta compartida entre dos partes (consulte la Figura 1), a las que aquí nos referiremos como Alice y Bob. Alice comienza ejecutando KeyGen para generar una clave de encapsulación (pública) y una clave de decapsulación (privada). Al obtener la clave de encapsulación de Alice, Bob ejecuta el algoritmo Encaps ; esto produce la copia KB de Bob de la clave secreta compartida junto con un texto cifrado asociado. Bob envía el texto cifrado a Alice y Alice completa el proceso ejecutando el algoritmo Decaps utilizando su clave de decapsulación y el texto cifrado; este paso produce la copia KA de Alice de la clave secreta compartida.

Después de completar el proceso anterior, a Alice y Bob les gustaría concluir que sus resultados individuales satisfacen KA = KB y que este valor es una clave secreta compartida, aleatoria y segura. Sin embargo, estas propiedades sólo se mantienen bajo ciertos supuestos importantes, como se analiza en NIST SP 800-227 [1].

3.2 El esquema ML-KEM

ML-KEM es un mecanismo de encapsulación de claves basado en CRYSTALS-KYBER [4], un esquema que se describió inicialmente en [8]. La siguiente es una descripción breve e informal de la suposición computacional subyacente a ML-KEM y cómo se construye el esquema ML-KEM .

El supuesto computacional. La seguridad de ML-KEM se basa en la presunta dificultad de resolver el llamado problema de Aprendizaje de Módulos con Errores (MLWE) [9], una generalización del problema de Aprendizaje con Errores (LWE) introducido por Regev en 2005 [10]. La dureza del problema MLWE se basa en la supuesta dureza de ciertos problemas computacionales en redes de módulos [9]. De ahí el nombre del esquema ML-KEM.

En el problema LWE, la entrada es un conjunto de ecuaciones lineales aleatorias "ruidosas" en algunas variables secretas x $\mathbb{Z}_q^n$ y la tarea es recuperar x. El ruido en las ecuaciones es tal que los algoritmos estándar (por ejemplo, la eliminación gaussiana) son intratables. El problema LWE se presta naturalmente a aplicaciones criptográficas. Por ejemplo, si x se interpreta como una clave secreta, entonces se puede cifrar un valor de un bit muestreando una ecuación lineal aproximadamente correcta (si el valor del bit es cero) o una ecuación lineal que dista mucho de ser correcta (si el valor del bit es cero). el valor es uno). Es plausible que sólo una parte en posesión de x pueda distinguir estos dos casos. Luego, el cifrado se puede delegar a otra parte mediante la publicación de una gran colección de ecuaciones lineales ruidosas, que la parte que cifra puede combinar adecuadamente . El resultado es un esquema de cifrado asimétrico.

A alto nivel, el problema MLWE plantea la misma tarea que LWE pero con $\mathbb{Z}_q^n$ reemplazado con el módulo R_q^k construido tomando el producto cartesiano k veces mayor de un cierto anillo polinómico R_q para algún entero k > 1. En particular, el secreto es ahora un elemento x del módulo R_q^k.

La construcción ML-KEM . A alto nivel, la construcción del ML-KEM se desarrolla en dos pasos. Primero, la idea mencionada anteriormente se utiliza para construir un esquema de cifrado de clave pública a partir del problema MLWE. En segundo lugar, este esquema de cifrado de clave pública se convierte en un mecanismo de encapsulación de claves mediante la denominada transformada Fujisaki-Okamoto (FO) [11, 12]. Además de producir un KEM, la transformación FO también pretende proporcionar seguridad en un modelo de ataque adversario significativamente más general. Como resultado, se cree que ML-KEM satisface la llamada seguridad IND-CCA [1, 4, 13].

La especificación de los algoritmos ML-KEM en este estándar seguirá el patrón anterior. Específicamente, este estándar describirá primero un esquema de cifrado de clave pública llamado K-PKE y luego utilizará los algoritmos de K-PKE como subrutinas al describir los algoritmos de ML-KEM. La transformación criptográfica de K-PKE a ML-KEM es crucial para lograr una seguridad total.

El esquema K-PKE no es lo suficientemente seguro y no debe utilizarse como un esquema independiente (consulte la Sección 3.3).

Una característica notable de ML-KEM es el uso de la transformada de teoría de números (NTT). La NTT convierte un polinomio f R_q a una representación alternativa como un vector f de polinomios lineales. Aunque las representaciones NTT permiten una multiplicación rápida, se deben aplicar otras operaciones, como el redondeo y el muestreo, a las representaciones polinómicas estándar.

ML-KEM satisface las propiedades clave de corrección de KEM y se conoce una prueba de seguridad teórica asintótica (en un determinado modelo heurístico) [4]. Cada uno de los conjuntos de parámetros de ML-KEM viene con una fortaleza de seguridad asociada, que se estimó en base al criptoanálisis actual (consulte la Sección 7 para obtener más detalles).

Conjuntos de parámetros y algoritmos. Recuerde que un KEM consta de algoritmos KeyGen, Encaps y Decaps, junto con una colección de conjuntos de parámetros. En el caso de ML-KEM, los tres algoritmos antes mencionados son:

- ML-KEM.KeyGen (Algoritmo 15);
- ML-KEM.Encaps (Algoritmo 16);
- ML-KEM.Decaps (Algoritmo 17).

Estos algoritmos se describen y analizan en detalle en la Sección 6.

ML-KEM viene equipado con tres conjuntos de parámetros:

- ML-KEM-512 (categoría de seguridad 1);
- ML-KEM-768 (categoría de seguridad 3);
- ML-KEM-1024 (categoría de seguridad 5).

Estos conjuntos de parámetros se describen y analizan en detalle en la Sección 7; las categorías de seguridad 1 a 5 se definen en el Apéndice A. Cada conjunto de parámetros asigna un valor numérico particular a cinco variables enteras: k, η_1, η_2, d_u y d_v. Los valores de estas variables en cada conjunto de parámetros se dan en la Tabla 2 de la Sección 7. Además de estos cinco parámetros variables, también hay dos constantes: $n = 256$ y $q = 3329$.

Fallos de decapsulación. Siempre que todas las entradas estén bien formadas, el procedimiento de establecimiento de claves de ML-KEM nunca fallará explícitamente. Específicamente, ML-KEM.Encaps y ML-KEM.Decaps Los algoritmos siempre generarán un valor con el mismo tipo de datos que una clave secreta compartida y nunca generarán un símbolo de error o falla. Sin embargo, es posible (aunque extremadamente improbable) que el proceso falle en el sentido de que Alice (a través de ML-KEM.Decaps) y Bob (a través de ML-KEM.Encaps) produzcan resultados diferentes, aunque ambos se comporten honestamente y no hay interferencia adversaria presente. En este caso, Alice y Bob claramente no lograron producir una clave secreta compartida. Este evento se llama falla de decapsulación. La probabilidad de falla de decapsulación se define como la probabilidad de que el proceso

1. (ek,dk) ← ML-KEM.KeyGen()

2. (c,K) ← ML-KEM.Encaps(ek)

3. K′ ← ML-KEM.Decaps(c,dk)

da como resultado K = K′ (es decir, la clave encapsulada es diferente de la clave decapsulada). Las estimaciones de la probabilidad (o tasa) de falla de decapsulación para cada uno de los conjuntos de parámetros ML-KEM se dan en la Tabla 1 (ver [4]).

Tabla 1. Tasas de falla de decapsulación para ML-KEM

Conjunto de parámetros	Tasa de fallo de decapsulación
ML-KEM-512	2-139
ML-KEM-768	2-164
ML-KEM-1024	2-174

Una nota sobre la terminología de las claves. Un KEM implica tres tipos diferentes de claves: claves de encapsulación, claves de decapsulación y claves secretas compartidas. ML-KEM se basa en el esquema de cifrado de clave pública K-PKE, y K-PKE tiene dos tipos de claves adicionales: claves de cifrado y claves de descifrado. En la literatura, las claves de encapsulación y las claves de cifrado a veces se denominan "claves públicas", mientras que las claves de decapsulación y descifrado a veces se pueden denominar "claves privadas". Para reducir la confusión, este estándar no utilizará los términos "clave pública" y "clave privada". En su lugar, nos referiremos a las claves utilizando los términos más específicos anteriores (es decir, clave de encapsulación, clave de decapsulación, clave de cifrado, clave de descifrado o clave secreta compartida).

3.3 Requisitos para implementaciones ML-KEM

Esta sección describe varios requisitos para implementar los algoritmos de ML-KEM. Los requisitos para utilizar ML-KEM en aplicaciones específicas se dan en NIST SP 800-227 [1].

K-PKE es sólo un componente. El esquema de cifrado de clave pública K-PKE descrito en la Sección 5 no se utilizará como esquema criptográfico independiente. En cambio, los algoritmos que componen K-PKE solo pueden usarse como subrutinas en los algoritmos de ML-KEM. En particular, los algoritmos K -PKE.KeyGen (Algoritmo 12), K-PKE.Encrypt (Algoritmo 13) y K-PKE.Decrypt (Algoritmo 14) no están aprobados para su uso como esquema de cifrado de clave pública.

Implementaciones equivalentes. Cada uno de los tres algoritmos de nivel superior (es decir, ML-KEM.KeyGen, ML-KEM.Encaps y ML-KEM.Decaps) define una operación matemática particular, asignando cualquier entrada dada a una salida correspondiente. Por ejemplo, la operación definida por el algoritmo ML-KEM.Encaps toma una matriz de bytes como entrada y produce dos matrices de bytes como salida.

En este estándar, las tres operaciones definidas por ML-KEM.KeyGen, ML-KEM.Encaps y ML-KEM.Decaps se describen utilizando secuencias particulares de pasos computacionales. Una implementación conforme puede reemplazar cada una de estas secuencias con una secuencia diferente de pasos, siempre que la operación resultante sea un proceso equivalente al especificado en este estándar.

Por ejemplo, una implementación conforme de la operación de encapsulación debe tener la propiedad de que, para cualquier conjunto de parámetros y cualquier matriz de bytes de entrada ek, la distribución de las matrices de bytes de salida sea idéntica a la distribución ML-KEM.Encaps(ek) como se especifica en este estándar.

Uso aprobado de la clave secreta compartida. La salida de los algoritmos de encapsulación y decapsulación de ML-KEM es siempre un valor de 256 bits. En condiciones apropiadas (ver arriba; ver también NIST SP 800-227 [1]), esta salida es una clave secreta compartida K. Esta clave secreta compartida K se puede usar directamente como clave para criptografía simétrica. Cuando sea necesaria la derivación de claves, las claves simétricas finales se derivarán de esta clave secreta compartida K de 256 bits de una manera aprobada , como se especifica en NIST SP 800-108 [14].

Generación de aleatoriedad. Tres algoritmos de este estándar requieren la generación de aleatoriedad: K-PKE.KeyGen, ML-KEM.KeyGen y ML-KEM.Encaps. En pseudocódigo, el paso en el que se genera esta aleatoriedad se indica mediante una declaración de pseudocódigo de la forma $m \xleftarrow{\$} \mathbb{B}^{32}$. Se debe generar una nueva cadena de bytes aleatorios para cada invocación de este tipo. Estos bytes aleatorios se generarán utilizando un RBG aprobado , según lo prescrito en NIST SP 800-90A, NIST SP 800-90B y NIST SP 800-90C [15, 16, 17]. Además, el RBG utilizado deberá tener una seguridad de al menos 128 bits para ML-KEM-512, al menos 192 bits para ML-KEM-768 y al menos 256 bits para ML-KEM-1024.

Validación de entrada. Los algoritmos ML-KEM.Encaps y ML-KEM.Decaps requieren validación de entrada. Los implementadores deberán garantizar que ML-KEM.Encaps y ML-KEM.Decaps solo se ejecuten en entradas validadas, como se describe en la Sección 6. Como se analizó anteriormente, los implementadores pueden optar por implementar los algoritmos de nivel superior (es decir, ML-KEM.Encaps , ML-KEM.Decaps o ML-KEM.KeyGen) utilizando cualquier proceso equivalente; la validación de los insumos se considera parte de este proceso. Una implementación conforme será equivalente a validar primero la entrada y luego ejecutar el algoritmo apropiado.

Destrucción de valores intermedios. Los datos utilizados internamente por los algoritmos KEM en pasos de cálculo intermedios podrían ser utilizados por un adversario para comprometer la seguridad. Por lo tanto, los ejecutores se asegurarán de que dichos datos intermedios se destruyan tan pronto como ya no sean necesarios.

Sin aritmética de punto flotante. Las implementaciones de ML-KEM no deben utilizar aritmética de punto flotante. Todos los pasos de división y redondeo de los algoritmos de ML-KEM se pueden realizar dentro del conjunto de números racionales.

4. Algoritmos auxiliares

4.1 Funciones criptográficas

Los algoritmos especificados en esta publicación requieren el uso de varias funciones criptográficas. Se creará una instancia de cada función mediante una función hash aprobada o una función de salida extensible (XOF) aprobada , como se describe a continuación. Las funciones hash y XOF relevantes se describen en detalle en FIPS 202 [7]. Se utilizarán de la siguiente manera.

SHA3-256 y SHA3-512 son funciones hash con entrada de longitud variable y salida de longitud fija. En este estándar, las invocaciones de estas funciones en una entrada M se indicarán con SHA3-256(M) y SHA3-512(M), respectivamente.

SHAKE128 y SHAKE256 son XOF con entrada y salida de longitud variable. Las invocaciones de estas funciones en una entrada M se indicarán de dos maneras diferentes, dependiendo de si se conoce la longitud de salida deseada ℓ (en bytes) en el momento de la invocación. Si se conoce ℓ en el momento de la invocación, la invocación se indicará como SHAKE128(M, ℓ) o SHAKE256(M, ℓ). Para SHAKE128, a veces no se conocerá la longitud de salida en el momento de la invocación; en esos casos, la invocación se indicará con SHAKE128(M) y la rutina de hash se comportará como un flujo de bytes que proporciona bytes pseudoaleatorios (realizando rondas de “compresión” adicionales [7]) hasta que no se necesiten más bytes.

Las funciones anteriores desempeñarán varios roles diferentes en los algoritmos especificados en este estándar. Será conveniente asignar una notación específica a cada uno de estos roles, como sigue.

Función pseudoaleatoria (PRF). La función PRF toma un parámetro η {2,3}, una entrada de 32 bytes y una entrada de 1 byte. Produce una salida de $(64 \cdot \eta)$ bytes. Se denotará por PRF : $\{2,3\} \times B^{32} \times B \to B^{64\eta}$, y se instanciará como

$$\mathrm{PRF}_\eta(s,b) := \mathrm{SHAKE256}(s \quad b, 64 \cdot \eta), \tag{4.1}$$

donde η {2,3}, s B^{32} y b B. Aquí, η solo se usa para especificar la longitud de salida deseada y no para realizar la separación de dominios. Tenga en cuenta que el parámetro de longitud de salida para SHAKE256 se especifica en bytes.

Función de salida extensible (XOF). La función XOF toma una entrada de 32 bytes y dos de 1 entradas de bytes. Produce una salida de longitud variable. Esta función se denotará por XOF : $B^{32} \times B \times B \to$ B , y se instanciará como

$$\mathrm{XOF}(\rho, i, j) := \mathrm{SHAKE128}(\rho \quad i \quad j), \tag{4.2}$$

donde ρ B^{32}, i B y j B. La función XOF solo se invocará para proporcionar un flujo de bytes pseudoaleatorios para el algoritmo de muestreo SampleNTT (Algoritmo 6). Como muestraNTT realiza un muestreo de rechazo, el número total de bytes necesarios no se conocerá en el momento en que se invoque XOF.

Tres funciones hash. La especificación también hará uso de tres instancias de función hash H, J y G, como se muestra a continuación.

Las funciones H y J toman cada una una entrada de longitud variable y producen una salida de 32 bytes. Se denotarán por H : B →B32 y J : B →B32, respectivamente, y se instanciarán como

$$H(s) := \text{SHA3-256}(s) \quad \text{y} \quad J(s) := \text{AGITAR256}(s,32) \tag{4.3}$$

donde s B .

La función G toma una entrada de longitud variable y produce dos salidas de 32 bytes. Se denotará por G : B →B32 × B32. Las dos salidas de G se denotarán, por ejemplo, por (a,b) ← G(c), donde a,b B32, c B y G(c) = a b. La función G se instanciará como

$$\text{GRAMO}(c) := \text{SHA3-512}(c). \tag{4.4}$$

4.2 Algoritmos generales

Esta sección especifica una serie de algoritmos que se utilizarán como subrutinas en los principales algoritmos de ML-KEM . Para una discusión sobre cómo interpretar el pseudocódigo de estos algoritmos, consulte la Sección 2.4.

4.2.1 Algoritmos de conversión y compresión

Esta sección especifica varios algoritmos para convertir entre matrices de bits, matrices de bytes y matrices de enteros módulo m. también especifica una determinada operación de compresión para números enteros módulo q, así como la correspondiente operación de descompresión.

Conversión entre bits y bytes. Los algoritmos 2 y 3 convierten entre matrices de bits y matrices de bytes. Las entradas de BitsToBytes y las salidas de BytesToBits son matrices de bits, y cada segmento de 8 bits representa un byte en orden little-endian.

Algoritmo 2 BitsToBytes(b)

Convierte una cadena de bits (de longitud múltiplo de ocho) en una matriz de bytes.

Entrada: matriz de bits b {0,1}8·ℓ .

Salida: matriz de bytes B Bℓ·

1: B ← (0,...,0)

2: para (i ← 0; i < 8ℓ; i++) 2i

3: ← B[i/8] +b[i] · mod 8 B[i/8]

4: fin para

5: volver B

Algoritmo 3 BytesABits(B)

Realiza lo inverso de BitsToBytes, convirtiendo una matriz de bytes en una matriz de bits.

Entrada: matriz de bytes B B^{ℓ}·

Salida: matriz de bits b $\{0,1\}^{8\cdot\ell}$.

```
para (i ← 0; i < ℓ; i++) para ( j ←
    0; j < 8; j++)
        b[8i+ j] ← B[i] mod 2
        B[yo] ← B[yo]/2
    terminar
para terminar
para regresar b
```

Compresión y descompresión. Recuerde que q = 3329 y observe que la longitud de bits de q es 12. Para d < 12, defina

$$\text{Comprimido} : \mathbb{Z}_q \longrightarrow \mathbb{Z}_{2^d} \tag{4.5}$$

$$x \longmapsto (2^d/q) \cdot x \; .$$

$$\text{Descomprimido} : \mathbb{Z}_{2^d} \longrightarrow \mathbb{Z}_q \tag{4.6}$$

$$y \longmapsto (q/2^d) \cdot y \; .$$

Tenga en cuenta que los tipos de entrada y salida de estas funciones son números enteros módulo m (consulte la discusión sobre tipos en la Sección 2.4). La división y el redondeo en el cálculo de las funciones anteriores se realizan en el conjunto de números racionales. No se deben utilizar cálculos de punto flotante .

De manera informal, Compress descarta bits de orden inferior de la entrada y Decompress agrega bits de orden inferior establecidos en cero. Estos algoritmos satisfacen dos propiedades importantes. Primero, la descompresión seguida de la compresión preserva la entrada, es decir, $\text{Compress}_d(\text{Decompress}_d(y)) = y$ para todo y $\mathbb{Z}_q$ y todo d < 12. Segundo, si d es grande (es decir, cercano a 12) , lo que significa que el El número de bits descartados es pequeño: la compresión seguida de la descompresión no altera significativamente el valor.

Específicamente,

$$[\text{Descomprimido}_d(\text{Comprimido}_d(x)) - x] \bmod^{\pm} q \leq q/2^{d+1} \tag{4.7}$$

para todo x $\mathbb{Z}_q$ y todo d < 12.

Codificación y decodificación. Los algoritmos ByteEncode (Algoritmo 4) y ByteDecode (Algoritmo 5) se utilizarán para la serialización y deserialización de matrices de números enteros módulo m. Todas las matrices serializadas tendrán una longitud de n = 256. ByteEncode_d serializa una matriz de enteros de d bits en una matriz de 32 · d bytes. ByteDecode_d realiza la operación de deserialización correspondiente, convirtiendo una matriz de 32 · d bytes en una matriz de enteros de d bits.

Para la siguiente discusión, es conveniente considerar ByteDecode y ByteEncode como conversiones entre números enteros y bits. (La conversión entre bits y bytes es sencilla y se realiza utilizando BitsToBytes y BytesToBits).

El rango válido de valores para el parámetro d es 1 ≤ d ≤ 12. Las matrices de bits se dividen en segmentos de d bits . En el caso de que 1 ≤ d ≤ 11, ByteDecoded convierte cada segmento de d bits de la entrada en un entero módulo 2d, y ByteEncoded realiza la operación inversa. En este caso la conversión es uno a uno.

El caso d = 12 se trata de manera diferente. En este caso, ByteEncode12 recibe números enteros módulo q como entrada, y ByteDecode12 produce números enteros módulo q como salida. ByteDecode12 convierte cada segmento de 12 bits de la entrada en un módulo entero 212 = 4096, y luego reduce el resultado en módulo q. Esta ya no es una operación uno a uno. De hecho, algunos segmentos de 12 bits podrían corresponder a un número entero mayor que q = 3329 pero menor que 4096; sin embargo, esto no puede ocurrir con las matrices producidas por ByteEncode12. Estos aspectos de ByteDecode12 y ByteEncode12 serán importantes al considerar la validación de la clave de encapsulación ML-KEM en la Sección 6.

Algoritmo 4 bytes codificados (F)

Codifica una matriz de enteros de d bits en una matriz de bytes, para 1 ≤ d ≤ 12.

Entrada: matriz de enteros F Z256, donde m = 2d si d < 12 y m = q si d = 12.
Salida: matriz de bytes B B32d.

```
para (i ← 0; i < 256; i++) a ←
    F[i] para                                   a    Z2d
    ( j ← 0; j < d; j++)
        b[i· d + j] ← a mod 2 a ←                b    {0,1}256·d
        (a−b[i · d + j])/2 final para 7:         nota a−b[i · d + j] siempre es par.
    fin para

B ← Bits a bytes (b)
volver B
```

Algoritmo 5 bytes decodificados (B)

Decodifica una matriz de bytes en una matriz de enteros de d bits, para 1 ≤ d ≤ 12.

Entrada: matriz de bytes B B32d.
Salida: matriz de enteros F Z256, donde m = 2d si d < 12 y m = q si d = 12.

```
b ← BytesABits(B)
para (i ← 0; i < 256; i++)
    F[yo] ← ∑d_{j=0}^{-1} b[i· d + j] · 2j mod m
fin para
volver F
```

4.2.2 Algoritmos de muestreo

Los algoritmos de ML-KEM requieren dos subrutinas de muestreo que se especifican en los algoritmos 6 y 7. Ambos algoritmos se pueden utilizar para convertir un flujo de bytes uniformemente aleatorios en una muestra de alguna distribución deseada. En este estándar, estos algoritmos se invocarán con un flujo de bytes pseudoaleatorios como entrada. De ello se deduce que el resultado será una muestra de una distribución que es computacionalmente indistinguible de la distribución deseada.

Muestreo uniforme de representaciones NTT. El algoritmo SampleNTT (Algoritmo 6) convierte un flujo de bytes en un polinomio en el dominio NTT. Si el flujo de entrada consta de bytes uniformemente aleatorios, entonces el resultado se extraerá uniformemente al azar de T_q. La salida es una matriz en Z^{256}

$_q$ que contiene los coeficientes del elemento muestreado de T_q (ver Sección 2.4).

Algoritmo 6 MuestraNTT(B)

Si la entrada es un flujo de bytes uniformemente aleatorios, la salida es un elemento uniformemente aleatorio de T_q.

Entrada: flujo de bytes B B .

Salida : matriz 1: a Z_q^{256} . los coeficientes del NTT de un polinomio

i ← 0

2: j ← 0 3:

mientras j < 256 haz d1

4: ← B[i] +256 ·(B[i+1] mod 16)

5: d2 ← B[i+1]/16 +16 ·B[i+2]

6: si d1 < q

7: entonces $\hat{\text{un}}$ Z^{256}_q

8: aˆ[j] ← d1

9: j ← j

10: +1 final si si d2 < q y j < 256

11: entonces

12: aˆ[j] ← d2

13: j ← j

14: +1 final si

i ← i+3 15: final

mientras 16 : regresar aˆ

Algoritmo 7 MuestraPolyCBDη(B)

Si la entrada es un flujo de bytes uniformemente aleatorios, genera una muestra de la distribución Dη(Rq).

Entrada: matriz de bytes B B64η.

Salida: matriz f Z^{256}_q. 1: los coeficientes del polinomio muestreado

b ← BytesABits(B)

2: para (i ← 0; i < 256; i++) x ←

3: $\sum_{j=0}^{\eta-1}$ b[2iη + j]

4: y ← $\sum_{j=0}^{\eta-1}$ b[2iη +η + j]

5: f [i] ← x−y mod q 6: fin de f Z^{256}_q

7: retorno f

Muestreo a partir de la distribución binomial centrada. ML-KEM hace uso de una distribución especial Dη(Rq) de polinomios en Rq con coeficientes pequeños. Estos polinomios a veces denominarse “errores” o “ruido”. La distribución está parametrizada por un número entero η {2,3}. Para muestrear un polinomio a partir de $D_\eta(R_q)$, cada uno de sus coeficientes se muestrea de forma independiente a partir de una determinada distribución binomial centrada (CBD) en $\mathbb{Z}_q$. El algoritmo SamplePolyCBD (Algoritmo 7) muestra la matriz de coeficientes de un polinomio f R_q según la distribución $D_\eta(R_q)$, siempre que su entrada sea un flujo de bytes uniformemente aleatorios.

4.3 La transformada de la teoría de números

La transformada de teoría de números (o NTT) puede verse como una versión exacta y especializada de la transformada discreta de Fourier. En el caso de ML-KEM, el NTT se utiliza para mejorar la eficiencia de la multiplicación en el anillo R_q. Recuerde que R_q es el anillo $\mathbb{Z}_q[X]/(X^n +1)$ formado por polinomios de la forma $f = f_0 + f_1X + \cdots + f_{255}X^{255}$ donde f_j $\mathbb{Z}_q$ para todo j, equipado con aritmética módulo $X^n +1$.

El anillo R_q es naturalmente isomorfo a otro anillo, denominado T_q, que es una suma directa de 128 extensiones cuadráticas de $\mathbb{Z}_q$. El NTT es un isomorfismo computacionalmente eficiente entre estos dos anillos. Al ingresar un polinomio f R_q, el NTT genera un elemento $\hat{f}$:= NTT(f) del anillo T_q, donde $\hat{f}$ se llama “representación NTT” de f . La propiedad de isomorfismo implica que

$$f \times_{R_q} g = \mathrm{NTT}^{-1}(\hat{f} \times_{T_q} \hat{g}), \tag{4.8}$$

donde $\times_{R_q}$ y $\times_{T_q}$ denotan multiplicación en R_q y T_q, respectivamente. Además, dado que T_q es un producto de 128 anillos, cada uno de los cuales consta de polinomios de grado uno, la operación $\times_{T_q}$ es mucho más eficiente que la operación $\times_{R_q}$. Por estas razones, el NTT se considera una parte integral de ML-KEM y no simplemente una optimización.

Como los anillos R_q y T_q tienen una estructura de espacio vectorial sobre $\mathbb{Z}_q$, el tipo de datos abstracto más natural para representar elementos de cualquiera de estos anillos es $\mathbb{Z}_q^n$. Por esta razón, la elección de la estructura de datos para las entradas y salidas de NTT y NTT-1 son matrices de longitud n de números enteros módulo q; Se entiende que estas matrices representan elementos de T_q o R_q, respectivamente (ver Sección 2.4). Ambos NTT y NTT−1 se puede calcular in situ. De hecho, los algoritmos 8 y 9 demuestran un medio eficiente para calcular NTT y NTT-1 in situ. Sin embargo, para mayor claridad en la comprensión de la distinción de los objetos algebraicos antes y después de la conversión, los algoritmos se escriben con entradas y salidas explícitas.

La estructura matemática de una NTT simple. Recuerde que, en ML-KEM, q es el primo $3329 = 2^8 \cdot 13 +1$ y $n = 256$. Hay 128 raíces unitarias primitivas 256 y ninguna raíz unitaria primitiva 512 en $\mathbb{Z}_q$. Tenga en cuenta que $\zeta = 17$ $\mathbb{Z}_q$ es una raíz primitiva número 256 del módulo unitario q. Así $\zeta^{128} \equiv -1$.

Defina BitRev7(i) como el número entero representado mediante la inversión de bits del valor de 7 bits sin signo que corresponde al número entero de entrada i {0,...,127}.

El polinomio $X^{256} +1$ se factoriza en 128 polinomios de grado 2 módulo q de la siguiente manera:

$$X^{256} +1 = \prod_{k=0}^{127} \left(X^2 - \zeta^{2\mathrm{BitRev}_7(k)+1}\right) . \tag{4.9}$$

Por lo tanto, $R_q := \mathbb{Z}_q[X]/(X^{256}+1)$ es isomorfo a una suma directa de 128 campos de extensión cuadrática de $\mathbb{Z}_q$, denotados T_q. En concreto, este anillo tiene la estructura

$$T_q := \bigoplus_{k=0}^{127} \mathbb{Z}_q[X]/\left(X^2 - \zeta^{2\mathsf{BitRev}_7(k)+1}\right). \tag{4.10}$$

Así, la representación NTT $\hat{f} \in T_q$ de un polinomio $f \in R_q$ es el vector que consta de residuos de grado uno correspondientes:

$$\hat{f} := \left(f \bmod (X^2 - \zeta^{2\mathsf{BitRev}_7(0)+1}), \ldots, f \bmod (X^2 - \zeta^{2\mathsf{BitRev}_7(127)+1})\right). \tag{4.11}$$

En los algoritmos siguientes, $\hat{f}$ se almacena como una matriz de 256 números enteros módulo q. Específicamente,

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

para i de 0 a 127.

Algoritmo 8 NTT(f)

Calcula la representación NTT $\hat{f}$ del polinomio dado $f \in R_q$.

Entrada: matriz $f \in \mathbb{Z}_q^{256}$. ▷ los coeficientes del polinomio de entrada

Salida: matriz $\hat{f} \in \mathbb{Z}_q^{256}$. ▷ los coeficientes de NTT del polinomio de entrada calculará

1: $\hat{f} \leftarrow f$ ▷ NTT in situ en una copia de la matriz de entrada

2: $k \leftarrow 1$

3: para (len ← 128; len ≥ 2; len ← len/2) para

4: (inicio ← 0; inicio < 256; inicio ← inicio+2 ·len)

5: zeta ← $\zeta^{\mathsf{BitRev}_7(k)} \bmod q$

6: k ← k +1

7: for (j ← inicio; j < inicio+len; j++) t ← zeta

8: $\cdot\ \hat{f}[j+\text{len}]$ $\hat{f}[j+\text{len}]$ ▷ pasos 8-10 realizados módulo q

9: $\leftarrow \hat{f}[j]-t$

10: $\hat{f}[j] \leftarrow \hat{f}[j]+t$

11: fin por

12: fin por

13: fin por

14: volver $\hat{f}$

Los algoritmos ML-KEM NTT. En el algoritmo 8 se describe un algoritmo para NTT . En el algoritmo 9 se describe un algoritmo para NTT inverso. Estos dos algoritmos están sobrecargados en este estándar. Primero, representan la transformación utilizada para mapear elementos de R_q a elementos de T_q (usando NTT) y viceversa (usando NTT^{-1}). En segundo lugar, representan la transformación coordinada de estructuras sobre esos anillos; Específicamente, asignan matrices/vectores con entradas en R_q a matrices/vectores con entradas en T_q (usando NTT) y viceversa (usando NTT^{-1}).

Algoritmo 9 NTT−1(f ˆ)

Calcula el polinomio f Rq correspondiente a la representación NTT dada f ˆ Tq.

Entrada: matriz f ˆ $\mathbb{Z}_q^{256}$. los coeficientes de la representación NTT de entrada

Salida: matriz f $\mathbb{Z}_q^{256}$. los coeficientes del NTT inverso de la entrada se calcularán in situ en una copia de la matriz de entrada

1: f ← f ˆ
2: k ← 127
3: para (len ← 2; len ≤ 128; len ← 2 · len) para
4: (inicio ← 0; inicio < 256; inicio ← inicio+2 ·len)
5: zeta ← ζ BitRev7(k) mod q
6: k ← k −1
7: para (j ← inicio; j < inicio+len; j++) t ← f [j]
8:
9: f [j] ← t + f [j +len] f [j pasos 9-10 realizados módulo q
10: +len] ← zeta ·(f [j +len]−t)
11: fin por
12: fin por
13: fin por 14:
f ← f · 3303 mod q 15: multiplica cada entrada por 3303 ≡128−1 mod q
retorno f

4.3.1 Multiplicación en el Dominio NTT

Como se analizó en la Sección 2.4, la suma y multiplicación escalar de elementos de Tq es sencilla: se puede hacer usando las operaciones aritméticas de coordenadas correspondientes en los conjuntos de coeficientes. Esta sección describe cómo realizar la operación del anillo restante (es decir, la multiplicación en Tq).

Recuerde de (4.11) que f ˆ Tq es un vector de polinomios cuadráticos de módulo de residuos uno de grado 1 . Algebraicamente, la multiplicación en el anillo Tq consiste en una multiplicación independiente en cada una de las 128 coordenadas respecto del módulo cuadrático de esa coordenada. Específicamente, la coordenada i-ésima en Tq del producto hˆ = f ˆ×Tq gˆ se determina mediante el cálculo

$$\hat{h}[2i] + \hat{h}[2i+1]X = (\hat{f}[2i] + \hat{f}[2i+1]X)(\hat{g}[2i] + \hat{g}[2i+1]X) \bmod (X^2 - \zeta^{2\mathrm{BitRev}_7(i)+1}). \quad (4.12)$$

Por tanto, se puede calcular el producto de dos elementos de Tq utilizando el algoritmo MultiplyNTTs (Algoritmo 10). Tenga en cuenta que MultiplyNTTs utiliza BaseCaseMultiply (Algoritmo 11) como subrutina. Como se analizó en la Sección 2.4, MultiplyNTTs permite realizar operaciones aritméticas algebraicas lineales con matrices y vectores con entradas en Tq.

Algoritmo 10 Multiplicar NTT($\hat{f}$,$\hat{g}$)

Calcula el producto (en el anillo T_q) de dos representaciones NTT.

Entrada: dos matrices $\hat{f}$ Z^{256} y $\hat{g}$ Z^{256}_q. los coeficientes de dos representaciones NTT los

Producción: Una matriz h Z_q^{256}. coeficientes del producto de las entradas

1: para (i ← 0; i < 128; i++)

2: fin[2i],h[2i+1]) ← BaseCaseMultiply($\hat{f}$[2i], $\hat{f}$[2i+1],$\hat{g}$[2i],$\hat{g}$[2i+1],ζ 2BitRev7(i)+1 (h) 3: de retorno

Algoritmo 11 BaseCaseMultiply(a0,a1,b0,b1, γ)

Calcula el producto de dos polinomios de grado uno con respecto a un módulo cuadrático.

Entrada: a_0,a_1,b_0,b_1 Zq. los coeficientes de $a_0 + a_1X$ y $b_0 + b_1X$

Entrada: γ Zq. el módulo es $X^2 - \gamma$

Salida: c_0,c_1 Zq. los coeficientes del producto de los dos polinomios pasos 1-2 realizados módulo q

1: $c_0 \leftarrow a_0 \cdot b_0 + a_1 \cdot b_1 \cdot \gamma$

2: $c_1 \leftarrow a_0 \cdot b_1 + a_1 \cdot b_0$

3: devuelve c_0, c_1

5. El esquema de componentes K-PKE

Esta sección describe el esquema de componentes K-PKE. Como se analizó en la Sección 3.3, K-PKE no está aprobado para su uso de forma independiente. Sirve únicamente como una colección de subrutinas para su uso en los algoritmos del esquema aprobado ML-KEM, como se describe en la Sección 6.

K-PKE consta de tres algoritmos:

1. Generación de claves (K-PKE.KeyGen);
2. Cifrado (K-PKE.Encrypt);
3. Descifrado (K-PKE.Decrypt).

Cuando se crea una instancia de K-PKE como parte de ML-KEM, K-PKE hereda el conjunto de parámetros seleccionado para ML-KEM. Cada conjunto de parámetros especifica valores numéricos para cada parámetro. Si bien n es siempre 256 y q es siempre 3329, los valores de los parámetros restantes k, η_1, η_2, d_u y d_v varían entre los tres conjuntos de parámetros. Los parámetros individuales y los conjuntos de parámetros se describen en la Sección 7.

Los algoritmos de esta sección no realizan ninguna validación de entrada. Esto se debe a que sólo se invocan como subrutinas de los principales algoritmos ML-KEM . Los algoritmos de ML-KEM realizan la validación de entradas según sea necesario; también garantizan que todas las entradas a los algoritmos K-PKE (invocadas como subrutinas) serán válidas.

Cada uno de los algoritmos de K-PKE a continuación va acompañado de una breve descripción informal en texto. Para simplificar, esta descripción se escribe en términos de vectores y matrices cuyas entradas son elementos de R_q. En el algoritmo real, la mayoría de los cálculos ocurren en el dominio NTT para mejorar la eficiencia de la multiplicación. Los vectores y matrices relevantes tendrán entonces entradas en T_q. La aritmética algebraica lineal con dichos vectores y matrices (ver, por ejemplo, la línea 19 de K-PKE.KeyGen) se realiza como se describe en las Secciones 2.4 y 4.3.1. La clave de cifrado y descifrado de K-PKE también se almacena en formato NTT.

5.1 Generación de claves K-PKE

El algoritmo de generación de claves K-PKE.KeyGen de K-PKE (Algoritmo 12) no requiere entrada, requiere aleatoriedad y genera una clave de cifrado ek_{PKE} y una clave de descifrado dk_{PKE}. Desde el punto de vista típico del cifrado de clave pública, la clave de cifrado puede hacerse pública, mientras que la clave de descifrado y la aleatoriedad deben permanecer privadas. Este será el caso también en el contexto de esta norma. De hecho, la clave de cifrado de K-PKE servirá como clave de encapsulación de ML-KEM (consulte ML-KEM.KeyGen a continuación) y, por lo tanto, puede hacerse pública; Mientras tanto, la clave de descifrado y la aleatoriedad de K-PKE.KeyGen deben permanecer privadas, ya que pueden usarse para realizar la desencapsulación en ML-KEM.

Descripción informal. La clave de descifrado de K-PKE.KeyGen es un vector s de longitud k de elementos de R_q, es decir, s R_q^k. En términos generales, s es un conjunto de variables secretas, mientras que la clave de cifrado es una colección de ecuaciones lineales "ruidosas" (A, $As+e$) en las variables secretas s. Las filas de la matriz A forman los coeficientes de la ecuación. Esta matriz se genera de forma pseudoaleatoria utilizando XOF, y solo la semilla se almacena en la clave de cifrado. El secreto s y el "ruido" e se muestrean del

Algoritmo 12 K-PKE.KeyGen()

Genera una clave de cifrado y una clave de descifrado correspondiente.

Salida: clave de cifrado ekPKE $\in \mathbb{B}^{384k+32}$.

Salida: clave de descifrado dkPKE $\in \mathbb{B}^{384k}$.

re $\xleftarrow{\$}$ B32 ▷ d tiene 32 bytes aleatorios (consulte la Sección 3.3)

(ρ ,σ) ← GRAMO(d) ▷ expandirse a dos semillas pseudoaleatorias de 32 bytes

N ← 0

para (i ← 0; i < k; i++) para ▷ generar matriz $\hat{A} \in (\mathbb{Z}_q^{256})^{k\times k}$

(j ← 0; j < k; j++)

  Aˆ [i, j] ← MuestraNTT(XOF(ρ ,i, j)) ▷ cada entrada de Aˆ uniforme en el dominio NTT

final para

final para

para (i ← 0; i < k; i++) ▷ generar s $\in (\mathbb{Z}_q^{256})^k$

  s[i] ← MuestraPolyCBDη1 (PRFη1 (σ ,N)) ▷ s[i] $\in \mathbb{Z}_q^{256}$ muestreado de CBD

N ← N +1 fin de

para (i ← 0; i < k; i++) ▷ generar e $\in (\mathbb{Z}_q^{256})^k$

  e[i] ← MuestraPolyCBDη1 (PRFη1 (σ ,N)) ▷ e[i] $\in \mathbb{Z}_q^{256}$ muestreado de CBD

N ← N +1 fin de

sˆ ← NTT(s) ▷ NTT se ejecuta k veces (una vez por cada coordenada de s)

eˆ ← NTT(e) ▷ NTT se ejecuta k veces

ˆt ← Aˆ ◦ sˆ +eˆ ▷ sistema lineal ruidoso en el dominio NTT

ekPKE ← ByteEncode12(ˆt) ∥ ρ ▷ ByteEncode12 se ejecuta k veces; incluir semilla para Aˆ

dkPKE ← ByteEncode12(sˆ) ▷ ByteEncode12 se ejecuta k veces

return (ekPKE,dkP KE)

Distribución binomial centrada utilizando aleatoriedad expandida desde una semilla vía PRF. Una vez generados A , s y e , se realiza el cálculo de la parte final t = As +e de la clave de cifrado.

En K-PKE.KeyGen, la elección del conjunto de parámetros afecta la longitud del secreto s (a través del parámetro k) y, como consecuencia, los tamaños del vector de ruido e y la matriz pseudoaleatoria A. La elección del conjunto de parámetros también afecta la distribución del ruido (a través del parámetro η1) utilizado para muestrear las entradas de s y e.

5.2 Cifrado K-PKE

El algoritmo de cifrado K-PKE.Encrypt de K-PKE (Algoritmo 13) toma una clave de cifrado ekPKE y un texto sin formato m como entrada, requiere aleatoriedad r y genera un texto cifrado c. Si bien muchos algoritmos especificados en este documento requieren aleatoriedad, solo la descripción de K-PKE.Encrypt interpreta esta aleatoriedad como parte de la entrada. Esto se debe a que ML-KEM necesitará invocar K-PKE.Encrypt con una elección específica de aleatoriedad (consulte el Algoritmo 16 para obtener más detalles).

Descripción informal. El algoritmo K-PKE.Encrypt comienza extrayendo el vector t y la semilla de la clave de cifrado. Luego, la semilla se expande para regenerar la matriz A, de la misma manera que se hizo en K-PKE.KeyGen. Si t y A se derivan correctamente de una clave de cifrado producida por K-PKE.KeyGen, entonces son iguales a sus valores correspondientes en K-PKE.KeyGen.

Algoritmo 13 K-PKE.Encrypt(ekPKE,m,r)

Utiliza la clave de cifrado para cifrar un mensaje de texto sin formato utilizando la aleatoriedad r.

Entrada: clave de cifrado ekPKE $\mathbb{B}^{384k+32}$.

Entrada: mensaje m $\mathbb{B}^{32}$.

Entrada: aleatoriedad de cifrado r $\mathbb{B}^{32}$.

Salida: texto cifrado c $\mathbb{B}^{32(d_u k+d_v)}$.

1: N ← 0 2:

ˆt ← ByteDecode12(ekPKE[0 : 384k])

3: ρ ← ekPKE[384k : 384k +32] 4: para (i ← 0; i < k; i++) para (j ← 0; j < k; j++) 5: Aˆ [i, j] ← SampleNTT(XOF (ρ ,yo,j))

6:

7: final para

8: final para

extraer semilla de 32 bytes de ekPKE

regenerar matriz A $(\mathbb{Z}_q^{256})^{k\times k}$

9: para (i ← 0; i < k; i++) — generar r $(\mathbb{Z}_q^{256})^k$

10: r[i] ← MuestraPolyCBDη1 (PRFη1 (r,N)) — r[i] $\mathbb{Z}_q^{256}$ muestreado de CBD

11: N ← N +1 12: final para 13: para (i ← 0; i < k; i++) — generar e1 $(\mathbb{Z}_q^{256})^k$

14: e1[i] ← MuestraPolyCBDη2 (PRFη2 (r,N)) — e1[i] $\mathbb{Z}_q^{256}$ muestreado de CBD

15: N ← N +1 16: fin de

17: e2 ← SamplePolyCBDη2 (PRFη2 (r,N)) 18: rˆ ← NTT(r) 19: u ← NTT−1(Aˆ ◦ rˆ) +e1 20: μ ← Descomprimir1(ByteDecode1(m)))

muestra e2 $\mathbb{Z}_q^{256}$ de CBD

NTT se ejecuta k veces

NTT−1 se ejecuta k veces

21: v ← NTT−1(ˆt ◦ rˆ) +e2 + μ 22: c1 ← ByteEncodedu (Compressdu (u)) 23: c2 ← ByteEncodedv (Compressdv (v))

codificar texto plano m en polinomio v.

ByteEncodedu se ejecuta k veces

24: devolver c ← (c1 c2)

Recuerde de la descripción de la generación de claves que el par (A,t = As +e) puede considerarse como un sistema de ecuaciones lineales ruidosas en las variables secretas s. Se puede generar una ecuación lineal ruidosa adicional en las mismas variables secretas (sin conocer s) eligiendo una combinación lineal aleatoria de las ecuaciones ruidosas en el sistema (A,t). Entonces se puede codificar información en el "término constante" (es decir, la entrada que es una combinación lineal de entradas de t) de dicha ecuación combinada. Esta información puede luego ser descifrada por una parte en posesión del art. Por ejemplo, se podría codificar un solo bit decidiendo si alterar significativamente o no el término constante, generando así una ecuación casi correcta (correspondiente al valor del bit descifrado de 0) o una ecuación que dista mucho de ser correcta (correspondiente al valor del bit descifrado de 1). En el caso de K-PKE, el término constante es un polinomio con 256 coeficientes, por lo que se puede codificar más información: un bit en cada coeficiente.

Para ello, K-PKE.Encrypt procede generando un vector r R_q^k y términos de ruido e1 R_q^k y e2 Rq, todos los cuales se muestrean a partir de la distribución binomial centrada utilizando pseudoaleatoriedad expandida (a través de PRF) a partir de la aleatoriedad de entrada r. Luego se calcula la "nueva ecuación ruidosa" que (hasta algunos detalles) se calcula mediante (u, v) ← (A r+e1,t r+e2). Luego se agrega una codificación µ apropiada del mensaje de entrada m al término t r +e2. Finalmente, el par (u, v) se comprime, se serializa en una matriz de bytes y se genera como texto cifrado.

5.3 Descifrado K-PKE

El algoritmo de descifrado K-PKE.Decrypt de K-PKE (Algoritmo 14) toma una clave de descifrado dkPKE y un texto cifrado c como entrada, no requiere aleatoriedad y genera un texto sin formato m.

Descripción informal. El algoritmo K-PKE.Decrypt comienza calculando la "ecuación ruidosa" (u,v) subyacente al texto cifrado c, como se explica en la descripción de K-PKE.Encrypt. Aquí se puede pensar en u como los coeficientes de la ecuación y en v como el término constante. Recuerde que la clave de descifrado dkPKE contiene el vector de variables secretas s. Por tanto, el algoritmo de descifrado puede utilizar la clave de descifrado para calcular el término constante verdadero v′ = s u y luego calcular v−v′ . El algoritmo de descifrado finaliza decodificando el mensaje de texto plano m de v−v′ y generando m.

Algoritmo 14 K-PKE.Decrypt(dkPKE, c)

Utiliza la clave de descifrado para descifrar un texto cifrado.

Entrada: clave de descifrado dkPKE B384k.

Entrada: texto cifrado c B32(duk+dv) ·

Salida: mensaje m B32. 1: c1

← c[0: 32duk]

2: c2 ← c[32duk : 32(duk +dv)]

3: u ← Descompressdv (ByteDecodedv (c2)) 4: v ← ByteDecodedu invocado k veces

Descompressdv (ByteDecodedv (c2))

5: sˆ ← ByteDecode12(dkPKE)

6: w ← v−NTT−1(sˆ ◦NTT(u)) 7: m ← NTT−1 y NTT invocados k veces

ByteEncode1(Compress1(w)) 8: devolver m decodificar texto plano m a partir del polinomio v

---

6. El mecanismo de encapsulación de claves ML-KEM

El esquema ML-KEM consta de tres algoritmos:

1. Generación de claves (ML-KEM.KeyGen)
2. Encapsulación (ML-KEM.Encaps)
3. Decapsulación (ML-KEM.Decaps)

Para crear una instancia de ML-KEM, se debe seleccionar un conjunto de parámetros, cada uno de los cuales está asociado con una compensación particular entre seguridad y rendimiento. Los tres conjuntos de parámetros posibles se denominan ML-KEM-512, ML-KEM-768 y ML-KEM-1024 y se describen en detalle en la Tabla 2 de la Sección 7. Cada conjunto de parámetros asigna valores numéricos específicos a los parámetros individuales n, q, k, η_1, η_2, d_u y d_v. Si bien n es siempre 256 y q es siempre 3329, los parámetros restantes varían entre los tres conjuntos de parámetros. Los implementadores deben asegurarse de que los tres algoritmos de ML-KEM enumerados anteriormente solo se invoquen con un conjunto de parámetros válido y que este conjunto de parámetros se seleccione de manera adecuada para la aplicación deseada. Además, los algoritmos ML-KEM.Encaps y ML-KEM.Decaps requieren validación de entradas, como se analiza a continuación.

6.1 Generación de claves ML-KEM

El algoritmo de generación de claves ML-KEM.KeyGen para ML-KEM (Algoritmo 15) no acepta entradas, requiere aleatoriedad y produce una clave de encapsulación y una clave de decapsulación. Si bien la clave de encapsulación se puede hacer pública, la clave de decapsulación debe permanecer privada.

Descripción informal. La subrutina principal de ML-KEM.KeyGen es el algoritmo de generación de claves de K-PKE (Algoritmo 12). La clave de encapsulación ML-KEM es simplemente la clave de cifrado de K-PKE. La clave de desencapsulación de ML-KEM se compone de la clave de descifrado de K-PKE, la clave de encapsulación, un hash de la clave de encapsulación y un valor pseudoaleatorio de 32 bytes. Este valor aleatorio se utilizará en el mecanismo de "rechazo implícito" del algoritmo de decapsulación (Algoritmo 17).

Algoritmo 15 ML-KEM.KeyGen()

Genera una clave de encapsulación y una clave de decapsulación correspondiente.

Salida: Clave de encapsulación ek B384k+32.
Salida: Clave de decapsulación dk B768k+96.

$ 1: z ⇐ B32 — z tiene 32 bytes aleatorios (consulte la Sección 3.3)
2: (ekPKE,dkPKE) ← K-PKE.KeyGen() 3: ek ← — generación de clave de ejecución para K-PKE
ekPKE 4: dk ← — La clave de encapsulación de KEM es solo la clave de cifrado de PKE
(dkPKE ek H(ek) z) 5: retorno — La clave de decaptación de KEM incluye la clave de descifrado de PKE
(ek,dk)

6.2 Encapsulación ML-KEM

El algoritmo de encapsulación ML-KEM.Encaps de ML-KEM (Algoritmo 16) acepta una clave de encapsulación como entrada, requiere aleatoriedad y genera un texto cifrado y una clave compartida.

Validación de entrada. Para validar una determinada entrada[1] eke en ML-KEM.Encaps, realice lo siguiente cheques.

1. (Verificación de tipo). Si eke no es una matriz de bytes de longitud 384k +32 para el valor de k especificado por el conjunto de parámetros correspondiente, la entrada no es válida.
2. (Verificación del módulo). Realice el cálculo ek ← ByteEncode12(ByteDecode12(eke)). Si la entrada no es válida. ek ≠ ek , (Ver Sección 4.2.1.)

Si cualquiera de las comprobaciones anteriores declara que la entrada no es válida, entonces ML-KEM.Encaps no se realizará con la entrada eke . En su lugar, se deben tomar medidas apropiadas para la aplicación para cancelar. Si se pasan las dos comprobaciones anteriores (es decir, ninguna de ellas declara que la entrada no es válida), entonces la entrada se considera válida y ML-KEM.Encaps se puede realizar con la entrada ek = eke . Es importante tener · en cuenta que la entrada anterior El proceso de validación no garantiza que eke sea una salida real de ML-KEM.KeyGen. De hecho, la capacidad de garantizar eso (sin utilizar la clave de decapsulación) violaría el supuesto de seguridad.

Recuerde que, como se analizó en la Sección 3.3, las implementaciones solo son necesarias para reproducir correctamente el comportamiento de entrada-salida de los algoritmos de nivel superior. En el caso de ML-KEM.Encaps, esto significa que una implementación puede realizar cualquier proceso equivalente a ejecutar las comprobaciones 1 y 2 anteriores y luego ejecutar el algoritmo 16. (Por ejemplo, la segunda verificación podría realizarse durante la ejecución de ByteDecode12 en la línea 2 de K-PKE.Encrypt.)

Algoritmo 16 ML-KEM.Encaps(ek)

Utiliza la clave de encapsulación para generar una clave compartida y un texto cifrado asociado.

Entrada validada: clave de encapsulación ek $\mathbb{B}^{384k+32}$.
Salida: clave compartida K $\mathbb{B}^{32}$.
Salida: texto cifrado c $\mathbb{B}^{32(d_u k+d_v)}$ ·

1: metro $\xleftarrow{\$}$ $\mathbb{B}^{32}$ ▷ m son 32 bytes aleatorios (consulte la Sección 3.3)
2: (K,r) ← G(m H(ek)) ▷ derivar la clave secreta compartida K y la aleatoriedad r
3: c ← K-PKE.Encrypt(ek,m,r) ▷ cifrar m usando K-PKE con aleatoriedad r
4: devolver (K, c)

Descripción informal. La subrutina principal de ML-KEM.Encaps es el algoritmo de cifrado de K-PKE, que se utiliza para cifrar un valor aleatorio m en un texto cifrado c. Una copia del secreto compartido K y la aleatoriedad utilizada durante el cifrado se derivan de m y la encapsulación.

[1]En discusiones sobre validación de entradas, la tilde en la notación indica que es posible que la entrada no esté formada correctamente, p.ej, eke para una entrada de clave de encapsulación candidata, a diferencia de ek para una entrada válida.

clave ek mediante hash. Específicamente, H se aplica a ek, y el resultado se concatena con my luego se aplica un hash usando G. El algoritmo se completa generando el texto cifrado c y el secreto compartido K.

6.3 Decapsulación ML-KEM

El algoritmo de decapsulación ML-KEM.Decaps de ML-KEM (Algoritmo 16) acepta una clave de decapsulación y un texto cifrado ML-KEM como entrada, no utiliza ninguna aleatoriedad y genera un secreto compartido.

Validación de entrada. Para validar un par determinado de entradas ce (texto cifrado candidato) y dke (clave de decapsulación candidata) en ML-KEM.Decaps, realice las siguientes comprobaciones.

1. (Verificación del tipo de texto cifrado). Si ce no es una matriz de bytes de longitud 32 (duk +dv) para los valores de du, dv y k especificados por el conjunto de parámetros relevante, la entrada no es válida.

2. (Verificación del tipo de clave de decapsulación). Si dke no es una matriz de bytes de longitud 768k +96 para el valor de k especificado por el conjunto de parámetros relevante, la entrada no es válida.

Si cualquiera de las comprobaciones anteriores declara que la entrada no es válida, entonces ML-KEM.Decaps no se realizará con la entrada (ce,dke). En su lugar, se deben tomar medidas apropiadas para la aplicación para cancelar. Si ambas comprobaciones pasan (es decir, ninguna declara que la entrada no sea válida), entonces la entrada es se considera válido y ML-KEM.Decaps se puede realizar con la entrada (c,dk) = (ce,dke).

Para algunas aplicaciones, puede ser apropiada una validación adicional de la clave de desencapsulación dke . Por ejemplo, en los casos en que dke fue generado por un tercero, es posible que los usuarios quieran asegurarse de que los cuatro componentes de dke tengan la relación correcta entre sí, como en la línea 4 de ML-KEM.KeyGen. En todos los casos, los implementadores validarán las entradas a ML-KEM.Decaps de una manera que sea apropiada para su aplicación.

Descripción informal. El algoritmo ML-KEM.Decaps comienza analizando los componentes de la clave de decapsulación dk de ML-KEM. Estos componentes son un par (clave de cifrado, clave de descifrado) para K-PKE, un valor hash h y un valor aleatorio z. Luego, la clave de descifrado de K-PKE se usa para descifrar el texto cifrado de entrada c para obtener un texto sin formato m . Luego, el algoritmo de desencapsulación vuelve a cifrar m′ y calcula una clave secreta compartida candidata K′ de la misma manera que se debería haber hecho en la encapsulación. . Específicamente, K′ y la aleatoriedad de cifrado r′ se calculan mediante el hash de m′ y la clave de cifrado de K-PKE, y se genera un texto cifrado c′ cifrando m′.

usando aleatoriedad r el ′. Finalmente, la decapsulación comprueba si el texto cifrado resultante c′ coincide texto cifrado proporcionado c. Si no es así, el algoritmo realiza un "rechazo implícito": el valor de K′ se cambia a un hash de c junto con el valor aleatorio z almacenado en la clave secreta ML-KEM (consulte la discusión sobre fallas de decapsulación en la Sección 3.2). En cualquier caso, la decapsulación genera la clave secreta compartida resultante K′ .

Algoritmo 17 ML-KEM.Decaps(c,dk)

Utiliza la clave de decapsulación para generar una clave compartida a partir de un texto cifrado.

Entrada validada: texto cifrado c B32(dk +dv).

validada: clave de decapsulación Salida: B768k+96 dk Entrada .

clave compartida K B32 1: ·

dkPKE ← dk[0 : 384k] 2: extraer (de la clave KEM decaps) la clave de descifrado PKE

ekPKE ← dk[384k : 768k +32] 3: h ← extraer la clave de cifrado PKE

dk[768k +32 : 768k +64] 4: z ← dk[768k extraer el hash de la clave de cifrado PKE

+64 : 768k +96] 5: m′ ← K- extraer el valor de rechazo implícito

PKE.Decrypt(dkPKE,c) descifrar el texto cifrado

K′ 6: (,r′) ← GRAMO(m′ h)

7: K¯ ← J(z c,32)

8: c′ ← K-PKE.Encrypt(ekPKE,m′ ,r′) 9: si c = c′ entonces volver a cifrar utilizando la aleatoriedad derivada r′

10: K′ ← K¯ si los textos cifrados no coinciden, "rechazar implícitamente"

11: terminar

si 12: devolver K′

7. Conjuntos de parámetros

ML-KEM está equipado con tres conjuntos de parámetros. Cada uno de los tres conjuntos de parámetros se compone de cinco parámetros individuales: k, η_1, η_2, du y dv. También hay dos constantes: n = 256 y q = 3329. La siguiente es una descripción breve e informal de los roles que desempeñan los parámetros variables en los algoritmos de K-PKE (y por lo tanto en ML-KEM). Consulte la Sección 5 para obtener más detalles.

- El parámetro k determina las dimensiones de los vectores s y e en K-PKE.KeyGen, así como las dimensiones de la matriz $\hat{A}$ y los vectores r, e1 y e2 en K-PKE.Encrypt.
- El parámetro η_1 es necesario para especificar la distribución para generar los vectores sye en K -PKE.KeyGen y el vector r en K-PKE.Encrypt.
- El parámetro η_2 es necesario para especificar la distribución para generar los vectores e1 y e2 en K-PKE.Encrypt.
- Los parámetros du y dv sirven como parámetros y entradas para las funciones Comprimir, Descomprimir, ByteEncode y ByteDecode en K-PKE.Encrypt y K-PKE.Decrypt.

Este estándar aprueba los conjuntos de parámetros que figuran en la Tabla 2. Cada conjunto de parámetros está asociado con una fuerza de seguridad requerida para la generación de aleatoriedad (consulte la Sección 3.3). Los tamaños de las claves ML-KEM y los textos cifrados para cada conjunto de parámetros se resumen en la Tabla 3.

	norte	q	k	η_1	η_2	du	dv	fuerza RBG requerida (bits)
ML-KEM-512	256	3329	2	3	2	10	4	128
ML-KEM-768	256	3329	3	2	2	10	4	192
ML-KEM-1024	256	3329	4	2	2	11	5	256

Tabla 2. Conjuntos de parámetros aprobados para ML-KEM

	clave de encapsulación	clave de decapsulación	texto cifrado	clave secreta compartida
ML-KEM-512	800	1632	768	32
ML-KEM-768	1184	2400	1088	32
ML-KEM-1024	1568	3168	1568	32

Tabla 3. Tamaños (en bytes) de claves y textos cifrados de ML-KEM

También se puede decir que el nombre de un conjunto de parámetros denota un KEM (sin parámetros). Específicamente, ML-KEM-x se puede utilizar para indicar el KEM sin parámetros que resulta de crear una instancia del esquema ML-KEM con el conjunto de parámetros ML-KEM-x.

Los tres conjuntos de parámetros incluidos en la Tabla 2 fueron diseñados para cumplir con ciertas categorías de resistencia de seguridad definidas por el NIST en su convocatoria de propuestas original [4, 18]. Estas categorías de fortaleza de seguridad se explican con más detalle en el Apéndice A.

Con este enfoque, la fortaleza de la seguridad no se describe con un solo número, como "128 bits de seguridad". En cambio, se afirma que cada conjunto de parámetros ML-KEM es al menos tan seguro como un conjunto de parámetros genérico.

cifrado de bloque con un tamaño de clave prescrito o una función hash genérica con una longitud de salida prescrita. Más precisamente, se afirma que los recursos computacionales necesarios para descifrar ML-KEM son mayores o iguales a los recursos computacionales necesarios para descifrar el cifrado de bloque o la función hash, cuando estos recursos computacionales se estiman utilizando cualquier modelo de cálculo realista. Los diferentes modelos de cálculo pueden ser más o menos realistas y, en consecuencia, conducir a estimaciones más o menos precisas de la solidez de la seguridad. Algunos modelos comúnmente estudiados se analizan en [19].

Concretamente, se afirma que ML-KEM-512 está en la categoría de seguridad 1, ML-KEM-768 se afirma que está en la categoría de seguridad 3 y ML-KEM-1024 se afirma que está en la categoría de seguridad 5. Para una discusión adicional sobre la fortaleza de seguridad de los criptosistemas basados en MLWE, ver [4].

Seleccionar un conjunto de parámetros apropiado. Al establecer inicialmente protecciones criptográficas para los datos, se debe utilizar el conjunto de parámetros más sólido posible . Esto tiene una serie de ventajas, incluida la reducción de la probabilidad de costosas transiciones a conjuntos de parámetros de mayor seguridad en el futuro. Al mismo tiempo, cabe señalar que algunos conjuntos de parámetros pueden tener efectos adversos en el rendimiento de la aplicación correspondiente (por ejemplo, el algoritmo puede ser inaceptablemente lento).

NIST recomienda utilizar ML-KEM-768 como conjunto de parámetros predeterminado, ya que proporciona un gran margen de seguridad a un costo de rendimiento razonable. En los casos en los que esto no sea práctico o cuando se requiera una seguridad aún mayor, se pueden utilizar otros conjuntos de parámetros.

Referencias

[1] NIST. Publicación especial 800-227: Recomendaciones para mecanismos de encapsulación de claves, 2024.

[2] Elaine B. Barker, Lily Chen, Allen L. Roginsky, Apostol Vassilev y Richard Davis. Recomendación para esquemas de establecimiento de claves por pares utilizando criptografía de logaritmos discretos . Informe técnico, publicación especial 800-56A Revisión 3, Departamento de Comercio de EE. UU., Washington, DC, abril de 2018.

[3] Elaine B. Barker, Lily Chen, Allen L. Roginsky, Apostol Vassilev, Richard Davis y Scott Simon. Recomendación para el establecimiento de claves por pares utilizando criptografía de factorización de enteros. Informe técnico, publicación especial 800-56B Revisión 2, Departamento de Comercio de EE. UU., Washington, DC, marzo de 2019.

[4] Robert Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler y Damien Stehlé. Especificaciones del algoritmo CRYSTALS-Kyber y documentación de respaldo. Presentación de tercera ronda al proceso de estandarización de criptografía poscuántica del NIST, 2020. https://csrc.nist.gov/proj ects/criptografía-post-cuántica/presentaciones de la ronda 3.

[5] Equipo de presentación de CRYSTALS-Kyber. "Discusión sobre la transformación FO modificada de Kyber". Publicación del foro en pqc-forum, disponible en https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/WFRDl8DqYQ4 , 2023.

[6] Equipo de presentación de CRYSTALS-Kyber. "Decisiones Kyber, parte 2: Transformación FO". Publicación del foro en pqc-forum, disponible en https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/C0D3 W1KoINY/m/99kIvydoAwAJ, 2023.

[7] Instituto Nacional de Normas y Tecnología. Estándar SHA-3: hash basado en permutaciones y funciones de salida extensible. (Departamento de Comercio de EE. UU., Washington, DC), Publicación de estándares federales de procesamiento de información (FIPS) 202, agosto de 2015. https: //doi.org/10.6028/NIST.FIPS.202.

[8] Joppe Bos, Léo Ducas, Eike Kiltz, T Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler y Damien Stehlé. CRYSTALS-Kyber: un KEM basado en celosía de módulo seguro CCA . En 2018 Simposio Europeo IEEE sobre Seguridad y Privacidad (EuroS&P), páginas 353–367, 2018.

[9] Adeline Langlois y Damien Stehlé. Reducciones del peor de los casos al promedio para celosías de módulos. Diseños, códigos y criptografía, 75(3):565–599, 2015.

[10] Oded Regev. Sobre celosías, aprendizaje con errores, códigos lineales aleatorios y criptografía. En Actas del trigésimo séptimo simposio anual de ACM sobre teoría de la computación, STOC '05, páginas 84–93, Nueva York, NY, EE. UU., 2005. Asociación de Maquinaria de Computación.

[11] Eiichiro Fujisaki y Tatsuaki Okamoto. Integración segura de asimétrico y simétrico. esquemas de cifrado. Revista de criptología, 26:80–101, 2013.

[12] Dennis Hofheinz, Kathrin Hövelmanns y Eike Kiltz. Un análisis modular de la transformación Fujisaki-Okamoto. En Yael Kalai y Leonid Reyzin, editores, Theory of Cryptography, páginas 341–371, Cham, 2017. Springer International Publishing.

[13] Jonathan Katz y Yehuda Lindell. Introducción a la criptografía moderna. Chapman y Salón/CRC, tercera edición, 2020.

[14] Lirio Chen. Recomendación para la derivación de claves utilizando funciones pseudoaleatorias. (Instituto Nacional de Estándares y Tecnología, Gaithersburg, MD), Publicación especial del NIST (SP) 800-108 Rev.1, agosto de 2022. https://doi.org/10.6028/NIST.SP.800-108r1.

[15] Elaine B. Barker y John M. Kelsey. Recomendación para la generación de números aleatorios utilizando generadores deterministas de bits aleatorios. (Instituto Nacional de Estándares y Tecnología, Gaithersburg, MD), Publicación especial (SP) del NIST 800-90A, Rev. 1, junio de 2015. https: //doi.org/10.6028/NIST.SP.800-90Ar1.

[16] Meltem Sönmez Turan, Elaine B. Barker, John M. Kelsey, Kerry A. McKay, Mary L. Baish y Mike Boyle. Recomendación para las fuentes de entropía utilizadas para la generación de bits aleatorios. (Instituto Nacional de Estándares y Tecnología, Gaithersburg, MD), Publicación especial (SP) del NIST 800-90B, enero de 2018. https:// doi.org/10.6028/NIST.SP.800-90B.

[17] Elaine B. Barker, John M. Kelsey, Kerry McKay, Allen Roginsky y Meltem Sönmez Turan. Recomendación para construcciones de generadores de bits aleatorios (RBG). (Instituto Nacional de Estándares y Tecnología, Gaithersburg, MD), Publicación especial (SP) 800-90C (Tercer borrador público) del NIST, septiembre de 2022. https://csrc.nist.gov/publications/detail/sp/800-90c/borrador.

[18] Instituto Nacional de Normas y Tecnología. Requisitos de presentación y criterios de evaluación para el proceso de estandarización de la criptografía poscuántica, 2016. https:// csrc.nist. gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals- fnal-dec-2016.pdf.

[19] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Yi-Kai Liu, Carl Miller, Dustin Moody, René Peralta, Ray Perlner, Angela Robinson y Daniel Smith-Tone. Informe de estado de la tercera ronda del proceso de estandarización de la criptografía poscuántica del NIST. Informe técnico NIST Interagencial o Informe interno (IR) 8413, Instituto Nacional de Estándares y Tecnología, Gaithersburg, MD, julio de 2022.

[20] Samuel Jaques, Michael Naehrig, Martin Roetteler y Fernando Virdia. Implementación de oráculos Grover para búsqueda de claves cuánticas en AES y LowMC. En Anne Canteaut y Yuval Ishai, editores, Advances in Cryptology – EUROCRYPT 2020, páginas 280–310, Cham, 2020. Publicaciones internacionales Springer.

[21] Lov K. Grover. Un algoritmo rápido de mecánica cuántica para la búsqueda de bases de datos. En procedimientos del Vigésimo Octavo Simposio Anual ACM sobre Teoría de la Computación, STOC '96, páginas 212–219, Nueva York, NY, EE. UU., 1996. Asociación de Maquinaria de Computación.

Apéndice A: Categorías de resistencia de la seguridad

El NIST comprende que existen importantes incertidumbres al estimar las fortalezas de seguridad de los criptosistemas poscuánticos. Estas incertidumbres provienen de dos fuentes: primero, la posibilidad de que se descubran nuevos algoritmos cuánticos, lo que conducirá a nuevos ataques criptoanalíticos; y segundo, nuestra capacidad limitada para predecir las características de rendimiento de las futuras computadoras cuánticas, como su costo, velocidad y tamaño de memoria.

Para abordar estas incertidumbres, el NIST propuso el siguiente enfoque en su convocatoria de propuestas original [18]. En lugar de definir la fortaleza de un algoritmo utilizando estimaciones precisas del número de “bits de seguridad”, el NIST definió una colección de categorías amplias de fortaleza de seguridad. Cada categoría se define mediante una primitiva de referencia comparativamente fácil de analizar, cuya seguridad servirá como base para una amplia variedad de métricas que el NIST considera potencialmente relevantes para la seguridad práctica. Se puede crear una instancia de un criptosistema determinado utilizando diferentes conjuntos de parámetros para poder encajar en diferentes categorías. Los objetivos de esta clasificación son:

- Facilitar comparaciones significativas de rendimiento entre varios algoritmos poscuánticos garantizando, en la medida de lo posible, que los conjuntos de parámetros que se comparan proporcionen una seguridad comparable.
- Para permitir que NIST tome decisiones futuras prudentes sobre cuándo realizar la transición a claves más largas
- Ayudar a los remitentes a tomar decisiones consistentes y sensatas con respecto a qué primitivas simétricas usar en los mecanismos de relleno u otros componentes de sus esquemas que requieren criptografía simétrica.
- Comprender mejor las compensaciones entre seguridad y rendimiento involucradas en un enfoque de diseño determinado.

De acuerdo con el segundo y tercer objetivo anteriores, el NIST basó su clasificación en la gama de fortalezas de seguridad que ofrecen los estándares existentes del NIST en criptografía simétrica, que el NIST espera que ofrezca una resistencia significativa al criptoanálisis cuántico. En particular, NIST definió una categoría separada para cada uno de los siguientes requisitos de seguridad (enumerados en orden creciente):

1. Cualquier ataque que viole la definición de seguridad relevante debe requerir recursos computacionales comparables o mayores que los necesarios para la búsqueda de claves en un cifrado de bloque con una clave de 128 bits (por ejemplo, AES-128).
2. Cualquier ataque que viole la definición de seguridad relevante debe requerir recursos computacionales comparables o mayores que los necesarios para la búsqueda de colisiones en una función hash de 256 bits (por ejemplo, SHA-256/SHA3-256).
3. Cualquier ataque que viole la definición de seguridad relevante debe requerir recursos computacionales comparables o mayores que los necesarios para la búsqueda de claves en un cifrado de bloque con una clave de 192 bits (por ejemplo, AES-192).
4. Cualquier ataque que viole la definición de seguridad relevante debe requerir recursos computacionales comparables o mayores que los necesarios para la búsqueda de colisiones en una función hash de 384 bits (por ejemplo, SHA-384/SHA3-384).
5. Cualquier ataque que rompa la definición de seguridad relevante debe requerir recursos computacionales

comparables o superiores a los necesarios para la búsqueda de claves en un cifrado de bloque con una clave de 256 bits (por ejemplo, AES-256).

Tabla 4. Categorías de fortaleza de seguridad del NIST

Categoría de seguridad	Tipo de ataque correspondiente	Ejemplo
1	Búsqueda de claves en cifrado de bloques con clave de 128 bits	AES-128
2	Búsqueda de colisiones en función hash de 256 bits	SHA3-256
3	Búsqueda de claves en cifrado de bloques con clave de 192 bits	AES-192
4	Búsqueda de colisiones en función hash de 384 bits	SHA3-384
5	Búsqueda de claves en cifrado de bloques con clave de 256 bits	AES-256

Aquí, los recursos computacionales se pueden medir usando una variedad de métricas diferentes (por ejemplo, número de operaciones elementales clásicas, tamaño del circuito cuántico). Para que un criptosistema satisfaga uno de los requisitos de seguridad anteriores, cualquier ataque debe requerir recursos computacionales comparables o superiores al umbral establecido, con respecto a todas las métricas que el NIST considera potencialmente relevantes para la seguridad práctica.

El NIST tiene la intención de considerar una variedad de métricas posibles, que reflejan diferentes predicciones sobre el desarrollo futuro de la tecnología de computación cuántica y clásica, y el costo de diferentes recursos informáticos (como el costo de acceder a cantidades extremadamente grandes de memoria).[2] El NIST también considerará aportes de la comunidad criptográfica con respecto a esta pregunta.

En una métrica de ejemplo proporcionada a los remitentes, el NIST sugirió un enfoque en el que los ataques cuánticos se restringen a un tiempo de ejecución fijo o a una profundidad del circuito. Llame a este parámetro MAXDEPTH. Esta restricción está motivada por la dificultad de ejecutar cálculos en serie extremadamente largos. Los valores plausibles para MAXDEPTH varían desde 240 puertas lógicas (el número aproximado de puertas que actualmente se espera que las arquitecturas de computación cuántica funcionen en serie en un año) hasta 264 puertas lógicas (el número aproximado de puertas que las arquitecturas de computación clásicas actuales pueden ejecutar en serie en un año). década), a no más de 296 puertas lógicas (el número aproximado de puertas que los qubits de escala atómica con tiempos de propagación de la luz podrían realizar en un milenio). La versión más básica de esta métrica de costos ignora los costos asociados con los bits o qubits que se mueven físicamente para que estén físicamente lo suficientemente cerca como para realizar operaciones de puerta. Esta simplificación puede resultar en una subestimación del costo de implementar cálculos con uso intensivo de memoria en hardware real.

La complejidad de los ataques cuánticos puede medirse entonces en términos del tamaño del circuito. Estos números se pueden comparar con los recursos necesarios para romper AES y SHA-3. Durante el proceso de estandarización poscuántica , NIST proporcionó las siguientes estimaciones para los recuentos de puertas clásicas y cuánticas[3] para la recuperación de claves óptima y los ataques de colisión en AES y SHA-3, respectivamente, donde

[2] Véase la discusión en [19, Apéndice B].

[3] Los tamaños de los circuitos cuánticos se basan en el trabajo de [20].

la profundidad del circuito está limitada a MAXDEPTH[4]].

Tabla 5. Estimaciones de recuentos de puertas clásicas y cuánticas para la recuperación óptima de claves y ataques de colisión en AES y SHA-3

Puertas cuánticas AES-128 2157/MAXDEPTH o puertas clásicas 2143
SHA3-256 2146 puertas clásicas
Puertas cuánticas AES-192 2221/MAXDEPTH o puertas clásicas 2207
SHA3-384 2210 puertas clásicas
Puertas cuánticas AES-256 2285/MAXDEPTH o 2272 puertas clásicas
SHA3-512 2274 puertas clásicas

Vale la pena señalar que las categorías de seguridad basadas en estas primitivas de referencia proporcionan sustancialmente más seguridad cuántica de lo que podría sugerir un análisis ingenuo. Por ejemplo, las categorías 1, 3 y 5 se definen en términos de cifrados en bloque, que pueden descifrarse utilizando el algoritmo de Grover [21], con una aceleración cuántica cuadrática. Sin embargo, el algoritmo de Grover requiere un cálculo en serie de larga duración , lo cual es difícil de implementar en la práctica. En un ataque realista, hay que ejecutar muchas instancias más pequeñas del algoritmo en paralelo, lo que hace que la aceleración cuántica sea menos dramática.

Finalmente, para los ataques que utilizan una combinación de computación clásica y cuántica, se puede utilizar una métrica de costos que califique las puertas cuánticas lógicas como varios órdenes de magnitud más caras que las puertas clásicas. Las arquitecturas de computación cuántica imaginadas actualmente suelen indicar que el costo por puerta cuántica podría ser miles de millones o billones de veces el costo por puerta clásica. Sin embargo, especialmente cuando se consideran algoritmos que afirman tener una alta seguridad (por ejemplo, equivalente a AES-256 o SHA-384), probablemente sea prudente considerar la posibilidad de que esta disparidad se reduzca significativamente o incluso se elimine.

[4] El NIST cree que las estimaciones anteriores son precisas para la mayoría de los valores de MAXDEPTH que son relevantes para su análisis de seguridad, pero las estimaciones anteriores pueden subestimar la seguridad de SHA para valores muy pequeños de MAXDEPTH y pueden subestimar la seguridad cuántica de AES para valores muy grandes. de MAXDEPTH.