centro criptológico nacional

Guía de Seguridad de las TIC
CCN-STIC 221

Guía de Mecanismos Criptográficos autorizados por el CCN



Noviembre de 2025

MINISTERIO DE DEFENSA

cpage.mpr.gob.es

Catálogo de Publicaciones de la Administración General del Estado
https://cpage.mpr.gob.es

Edita:

MINISTERIO DE DEFENSA
SUBSECRETARÍA DE DEFENSA
SECRETARÍA GENERAL TÉCNICA

Pº de la Castellana 109, 28046 Madrid
© Centro Criptológico Nacional, 2024

NIPO: 083-25-213-0
Fecha de Edición: noviembre de 2024

CCN centro criptológico nacional

CENTRO CRIPTOLOGICO NACIONAL
cn=CENTRO CRIPTOLOGICO NACIONAL, 2.5.4.97=VATES-S2800155J, ou=CENTRO CRIPTOLOGICO NACIONAL, o=CENTRO CRIPTOLOGICO NACIONAL, c=ES
2026.01.16 17:25:27 +01'00'

El grupo de investigación en Criptología y Seguridad de la Información (CiGSI) del Instituto de Tecnologías Físicas y de la Información "Leonardo Torres Quevedo" (ITEFI) del Consejo Superior de Investigaciones Científicas (CSIC) ha participado en la realización de esta guía.

LIMITACIÓN DE RESPONSABILIDAD

El presente documento se proporciona de acuerdo con los términos en él recogidos, rechazando expresamente cualquier tipo de garantía implícita que se pueda encontrar relacionada. En ningún caso, el Centro Criptológico Nacional puede ser considerado responsable del daño directo, indirecto, fortuito o extraordinario derivado de la utilización de la información y software que se indican incluso cuando se advierta de tal posibilidad.

AVISO LEGAL

Quedan rigurosamente prohibidas, sin la autorización escrita del Centro Criptológico Nacional, bajo las sanciones establecidas en las leyes, la reproducción parcial o total de este documento por cualquier medio o procedimiento, comprendidos la reprografía y el tratamiento informático, y la distribución de ejemplares del mismo mediante alquiler o préstamo públicos.

ÍNDICE

1 Introducción . . . 9
1.1 Objetivo de la Guía . . . 9
1.2 Mecanismos Criptográficos . . . 12
1.2.1 Algoritmos, Protocolos y Esquemas . . . 13
1.2.2 Tipos de Ataque . . . 15
1.2.3 Esquemas Simétricos y Asimétricos . . . 16
1.3 Complejidad de un Ataque y Niveles de Seguridad . . . 16
1.3.1 Complejidad de un Ataque . . . 17
1.3.2 Niveles de Seguridad . . . 18
1.4 Organización de la Guía . . . 20
2 Mecanismos Simétricos . . . 22
2.1 Primitivas Simétricas . . . 22
2.1.1 Cifradores en Flujo . . . 22
2.1.2 Cifradores en Bloque . . . 23
2.1.3 Funciones Resumen . . . 24
2.1.4 Funciones resumen con salida variable . . . 25
2.1.5 Compartición de Secretos . . . 26
2.2 Construcciones Simétricas . . . 27
2.2.1 Modos de operación en el Cifrado y Descifrado . . . 27
2.2.2 Cifrado de Disco Duro . . . 29
2.2.3 Códigos de Autenticación de Mensajes . . . 30
2.2.4 Esquemas Simétricos de Autenticación de Entidades . . . 33
2.2.5 Cifrado Autenticado . . . 34
2.2.6 Protección de las Claves . . . 36
2.2.7 Funciones de Derivación de Claves . . . 37
2.2.8 Mecanismos de Protección de Contraseñas . . . 38
2.2.9 Mecanismos de Combinación de Claves . . . 40
3 Mecanismos Asimétricos . . . 41
3.1 Primitivas Asimétricas . . . 41
3.1.1 Problema de la Factorización de Números Enteros (RSA) . . . 42
3.1.2 Problema del Logaritmo Discreto Multiplicativo . . . 43
3.1.3 Problema del Logaritmo Discreto Aditivo . . . 45
3.1.4 Otros Problemas Computacionalemente Difíciles . . . 48
3.2 Construcciones Asimétricas . . . 49
3.2.1 Esquemas de Cifrado Asimétrico . . . 49
3.2.2 Firmas Digitales . . . 51
3.2.3 Esquemas de Autenticación de Entidad Asimétrica . . . 55
3.2.4 Establecimiento de Claves y Encapsulación de Claves . . . 55
4 Protocolos Criptográficos . . . 60
4.1 TLS . . . 60
4.1.1 TLS Versión 1.3 . . . 65
4.1.2 TLS Versión 1.2 . . . 68

4.2 SSH . . . 71
4.2.1 Acuerdo de Clave . . . 72
4.2.2 Cifrado . . . 73
4.2.3 Integridad y Autenticidad en Origen . . . 74
4.2.4 Autenticación del Servidor y del Cliente . . . 75
4.3 IPSEC con IKEV2 . . . 75
4.3.1 Acuerdo de Claves . . . 76
4.3.2 Cifrado . . . 77
4.3.3 Integridad y Autenticación . . . 77
4.3.4 Funciones Pseudo-aleatorias . . . 78
4.3.5 Mecanismos de Autenticación . . . 78
5 Generadores de Números Aleatorios . . . 80
5.1 Generadores de Números Aleatorios . . . 80
5.2 Generadores Físicos de Números Aleatorios . . . 81
5.3 Generadores Deterministas de Números Aleatorios . . . 83
5.4 Generadores No Físicos de Números Realmente Aleatorios . . . 85
5.5 Generación de Números Aleatorios con una Distribución Específica . . . 87
6 Gestión de Claves . . . 89
6.1 Generación de Claves . . . 89
6.2 Almacenamiento y Transporte de Claves . . . 91
6.3 Uso de Claves . . . 91
6.4 Destrucción de Claves . . . 92
7 Autenticación de Personas . . . 93
7.1 Autenticación . . . 93
7.2 Procedimientos de Autenticación . . . 93
7.2.1 Limitación en el Número de Ensayos . . . 94
7.2.2 Limitación Temporal en el Número de Ensayos . . . 94
ANEXOS . . . 96
A Generación de Primos y Claves RSA . . . 97
A.1 Generación de Números Primos . . . 97
A.2 Test de Primalidad y de Pseudo-Primalidad . . . 99
A.2.1 Generación de Primos Probables (*Probable Primes*) . . . 100
A.3 Generación del Par de Claves de RSA . . . 103
A.4 Ataque ROCA al Algoritmo RSA . . . 104
B Criptografía Postcuántica . . . 106
B.1 La Amenaza Cuántica . . . 106
B.1.1 Convocatoria del NIST . . . 107
B.1.2 Seguridad . . . 109
B.2 Criptografía Basada en Retículos . . . 111
B.2.1 Retículos . . . 111
B.2.2 ML-KEM . . . 114
B.2.3 FRODOKEM . . . 125
B.2.4 ML-DSA . . . 127
B.3 Firmas Digitales Basadas en Funciones Resumen . . . 140

B.3.1 SLH-DSA 141
B.3.2 XMSS 154
B.4 Recomendaciones para la Resistencia Cuántica 157
B.4.1 Transición segura de la precuántica a la postcuántica 157
B.4.2 Modos de hibridación 160
B.4.2.1 Hibridación *CAT then KDF* 161
B.4.2.2 Hibridación *Cascade* 163
C Esquemas de Firma Digital Basados en el Problema Logaritmo Discreto 165
Glosario 182
Referencias 188

TABLAS

Tabla 2.1. Tipos autorizados de cifradores en bloque . . . 23
Tabla 2.2. Funciones resumen autorizadas . . . 25
Tabla 2.3. Primitivas XOF autorizadas . . . 26
Tabla 2.4. Compartición de secretos autorizada . . . 27
Tabla 2.5. Modos autorizados de cifrado simétrico . . . 28
Tabla 2.6. Modos autorizados de cifrado simétrico para cifrado de disco . . . 30
Tabla 2.7. MAC autorizados basados en cifradores en bloque y funciones resumen 31
Tabla 2.8. Tamaño de los protocolos de desafío-respuesta autorizados . . . 34
Tabla 2.9. Esquemas simétricos de cifrado autenticado autorizados . . . 35
Tabla 2.10. Esquemas de protección de claves autorizados . . . 37
Tabla 2.11. Caption without FN . . . 38
Tabla 2.12. Mecanismos de protección de contraseñas autorizados . . . 39
Tabla 2.13. Mecanismos de hibridación de claves autorizados . . . 40
Tabla 3.1. Tamaño de las Primitivas RSA autorizadas . . . 43
Tabla 3.2. Tamaño de las primitivas autorizadas del logaritmo discreto multiplicativo sobre un cuerpo finito . . . 44
Tabla 3.3. Esquemas autorizados para generar nuevos grupos . . . 44
Tabla 3.4. Parámetros autorizados para generar nuevos grupos . . . 44
Tabla 3.5. Curvas elípticas autorizadas . . . 47
Tabla 3.6. Esquema de cifrado asimétrico autorizado . . . 50
Tabla 3.7. Esquemas de firma digital autorizados . . . 52
Tabla 3.8. Esquemas de establecimiento de claves y de encapsulación de claves autorizados . . . 57
Tabla 4.1. Versiones del protocolo TLS autorizadas . . . 62
Tabla 4.2. Suites criptográficas autorizadas para el protocolo TLS 1.3 . . . 65
Tabla 4.3. Modos de clave precompartida recomendados para el protocolo TLS 1.3 65
Tabla 4.4. Grupos de Diffie-Hellman recomendados para el protocolo TLS 1.3 . 66
Tabla 4.5. Algoritmos de firma (cliente/servidor) recomendados para el protocolo TLS 1.3 . . . 67
Tabla 4.6. Algoritmos de firma (en certificados) recomendados para el protocolo TLS 1.3 . . . 67
Tabla 4.7. Suites criptográficas recomendadas para TLS 1.2 con un servidor que disponga de un certificado con la clave pública EC-DSA . . . 68
Tabla 4.8. Suites criptográficas heredadas para TLS 1.2 con un servidor que disponga de un certificado con la clave pública RSA . . . 68
Tabla 4.9. Suites criptográficas recomendadas para TLS 1.2 cuando no hay soporte ECC o modo de cifrado autenticado . . . 69
Tabla 4.10. Suites criptográficas recomendadas para TLS 1.2 con clave precompartida . . . 70
Tabla 4.11. Grupos de Diffie-Hellman recomendados para el protocolo TLS 1.2 . 71
Tabla 4.12. Algoritmos de firma recomendados para el protocolo TLS 1.2 . . . 71
Tabla 4.13. Funciones resumen para el protocolo TLS 1.2 . . . 71

Tabla 4.14. Versiones de acuerdos de clave para el protocolo SSH autorizadas . . 72
Tabla 4.15. Algoritmos de cifrado autorizados para el protocolo 74
Tabla 4.16. Funciones MAC autorizadas para el protocolo SSH 74
Tabla 4.17. Métodos de autenticación del servidor autorizados para el protocolo SSH 75
Tabla 4.18. Versiones autorizadas del protocolo IPsec 76
Tabla 4.19. Grupos de tipo DH y ECDH acordados por IKEv2 77
Tabla 4.20. Esquemas MAC y HMAC autorizados para IKEv2 y ESP 78
Tabla 4.21. Funciones pseudo-aleatorias autorizadas para IKEv2 78
Tabla 4.22. Esquemas de firma acordados para IKEv2 79
Tabla 5.1. Clases de generadores físicos de números aleatorios autorizados . . . 82
Tabla 5.2. Clases de generadores deterministas de números aleatorios autorizados 83
Tabla 5.3. Generadores deterministas de números aleatorios autorizados 84
Tabla 5.4. Clase de generador de números aleatorios ni físico ni determinista autorizado 86
Tabla 5.5. Esquemas autorizados de generación de números enteros aleatorios en un intervalo $[a, b]$ 88
Tabla 6.1. Métodos autorizados para la generación de claves genéricas 90
Tabla 7.1. Probabilidades máximas de falsa aceptación 94
Tabla A.1. Generación de primos por muestreo de rechazo 99
Tabla A.2. Test de primalidad probabilístico autorizado 100
Tabla A.3. Número de iteraciones del test de Miller-Rabin según la longitud en bits del candidato para $P_{obj} = 2^{-125}$ 102
Tabla A.4. Algoritmo autorizado para la generación claves RSA 103
Tabla B.1. Propuesta KEM seleccionada después de la tercera ronda y Primitiva matemática asociada 108
Tabla B.2. Propuestas a firmas seleccionadas después de la tercera ronda y Primitivas matemáticas asociadas 108
Tabla B.3. Candidatos a KEM para ser analizados en la cuarta ronda y Primitivas matemáticas asociadas 108
Tabla B.4. Propuesta KEM de interés para el CCN 109
Tabla B.5. Esquema general del establecimiento de claves mediante un KEM . . 115
Tabla B.6. Estimaciones de probabilidad de fallo de desencapsulado 115
Tabla B.7. Conjunto de parámetros aprobados para ML-KEM 125
Tabla B.8. Tamaño en bytes de las claves y textos cifrados para ML-KEM . . . 125
Tabla B.9. Conjunto de parámetros para ML-DSA 139
Tabla B.10. Categorías de seguridad establecidas por el NIST 140
Tabla B.11. Tamaños (en bytes) de las claves y de las especificaciones de la firma ML-DSA 140
Tabla B.12. Conjuntos de parámetros para SLH-DSA 152
Tabla B.13. Acuerdo de clave híbrido concatenado o *CAT then KDF* 161
Tabla B.14. Acuerdo de clave híbrido en cascada o *Cascade* 163

Tabla C.1. Curvas twisted de Edwards aprobadas para su uso en el esquema EdDSA173

1. Introducción

1.1. Objetivo de la Guía

1. El principal objetivo de este documento es especificar los mecanismos criptográficos que están reconocidos y aceptados por el Centro Criptológico Nacional (CCN), perteneciente al Centro Nacional de Inteligencia (CNI). Esta guía tiene como objetivo determinar de manera clara qué mecanismos son recomendados por el CCN para ser utilizados tanto por los desarrolladores que incorporan mecanismos criptográficos en sus productos como por los administradores que configuran los mecanismos criptográficos de los productos que emplean para asegurar sus sistemas.

2. En el caso de que alguna empresa u organismo deseara utilizar un mecanismo o dispositivo no incluido en esta Guía, podrá elevar su consulta al CCN, quien, en última instancia, será quien tome la decisión definitiva con relación al producto en cuestión.

3. Otros países de nuestro entorno también han publicado documentos o informes similares, con recomendaciones que tienen objetivos similares a los de esta Guía (véanse por ejemplo, [ANS20c], [ANS20a], [ACM1.3], [ANS21], [BSI22]). En España, el CCN [RD421/2004] tiene la responsabilidad y obligación de velar por la seguridad y protección de la información en el entorno criptográfico, por lo que esta Guía es de obligado cumplimiento. Esto es, los mecanismos criptográficos aquí recomendados tienen prioridad para su aplicación y uso en el territorio nacional, sean cuales sean las recomendaciones y sugerencias que otros países puedan publicar en sus correspondientes ámbitos de actuación.

4. El CCN es consciente de la globalización de todos los aspectos relacionados con la seguridad y protección de la información. Para ello el CCN ha tenido en cuenta el estado del arte de esta cuestión a nivel académico, las recomendaciones realizadas por otros países, así como las recomendaciones realizadas por organismos internacionales como SOG-IS (*Senior Officials Group-Information Systems Security*[1]).

5. Este documento pretende, por tanto, ser una guía enfocada a proporcionar las características de los productos que garantizan seguridad contra quienes pretendan atacar y vulnerar la confidencialidad, integridad, autenticación y no repudio de la información que deba ser protegida de accesos no permitidos. Además, el documento contiene consejos y recomendaciones relacionados con la posible implementación de diferentes mecanismos, de modo que puedan ser de utilidad para los desarrolladores y evaluadores nacionales.

[1] https://www.sogis.eu/

6. A lo largo de esta guía se incluirán diferentes cajas de texto, con fondos de diferentes colores, con información relevante y destacada. A continuación se muestra un ejemplo de cada una de ellas señalando el tipo de información que contiene.

7. [**Definición:**] estas cajas de texto con fondo verde contienen la definición del término que aparece en negrita y entre corchetes al inicio del mismo.

8. [**Recomendación:**] las cajas de texto con fondo amarillo contienen recomendaciones a tener en cuenta, pero sin que tengan un carácter obligatorio.

9. Nota 1 **(Consideraciones para desarrolladores y evaluadores.)** Este tipo de caja de texto con fondo gris hace referencia a los aspectos que se deben tener en cuenta a la hora de desarrollar o evaluar los diferentes mecanismos. Cada caja de texto puede considerarse como una nota que va numerada y está referenciada en la columna de ≪Notas≫ de las tablas que resumen las características de los mecanismos recomendados o heredados.

10. Por otra parte, es importante señalar que en esta Guía se consideran, inicialmente, dos tipos de productos criptográficos, fundamentalmente relacionados con su perdurabilidad en el tiempo, a tenor de la fortaleza o robustez que ofrecen, en tanto no se conozcan nuevos ataques, vulnerabilidades o debilidades que pongan en entredicho tal fortaleza. Por ello, se llevará a cabo una actualización permanente de esta Guía que tenga en cuenta los posibles ataques que se publiquen, ya sean de tipo criptoanalítico, por canal lateral e inducción de fallos, o por el desarrollo de la computación cuántica.

11. Los dos tipos de productos se clasifican en:

    - **Mecanismos recomendados**: son los mecanismos criptográficos que ofrecen una seguridad probaba de, al menos, 125 bits. Tal nivel de seguridad ha sido mostrado en publicaciones científicas (o son considerados estándares), de modo que el mismo está respaldado por argumentos científicos. Dicho de otro modo, no se conocen debilidades o amenazas en el estado del arte y la técnica que puedan poner tales argumentos en entredicho, ya sea considerando sus aspectos matemáticos o de ingeniería en seguridad criptográfica.

      Si alguno de los mecanismos recomendados que se consideran en esta Guía fuera objeto de algún ataque que pusiera en duda su seguridad, en futuras versiones de esta Guía, tal hecho será tenido en cuenta y se adoptarían las medidas oportunas en función de la calidad del ataque.

    - **Mecanismos heredados**: son aquellos mecanismos criptográficos de uso muy extendido que ofrecen un nivel de seguridad de, al menos, 100 bits,

de tal manera que su seguridad es solo aceptable a corto plazo. Esto es, son mecanismos que deben dejar de utilizarse en un corto plazo de tiempo porque el estado del arte ha demostrado que su garantía de seguridad se ha visto comprometida. Para estos mecanismos heredados, se ha considerado un periodo de validez. La principal razón para que el uso de estos mecanismos se mantenga durante determinado periodo de tiempo se debe a razones de compatibilidad dado que están implementados a gran escala y necesitan de un tiempo para ser sustituidos por otros más seguros (los recomendados).

Una vez concluido tal periodo de validez, el mecanismo se considerará obsoleto y no podrá ser utilizado al amparo de esta Guía. Así, la notación $H[2030]$ (H señala que es «heredado») significa que el periodo de validez concluye en diciembre de 2030, esto es, que tal mecanismo se acepta hasta esa fecha. Por su parte, la expresión $H[2030+]$ indica que el periodo de validez es mínimo, es decir, que la validez del mecanismo dado será, al menos, hasta finales de 2030; siendo posible que su validez sea ampliada en futuras versiones de esta Guía.

12. [**Mecanismos recomendados:**] son aquellos que ofrecen un nivel de seguridad criptográfica probada de, al menos, 125 bits.

13. [**Mecanismos heredados:**] son aquellos que ofrecen un nivel de seguridad criptográfica de entre 100 y 128 bits.El periodo de validez, por defecto, para los mecanismos heredados se establece en H[2030+].

14. Dado que no es posible establecer a priori durante cuánto tiempo un mecanismo criptográfico permanecerá siendo seguro, debido a la publicación de nuevos ataques o a la mejora en los tiempos de computación que puedan vulnerar los algoritmos en los que tales mecanismos se basan, esta Guía puede considerarse conservadora en el sentido de que primará la seguridad sobre cualquier otro aspecto. Por ello, los nuevos mecanismos criptográficos que puedan recomendarse en el futuro deberán utilizar los algoritmos y los tamaños de clave que hayan sido probados seguros.

15. Así, si el estado del arte lo aconseja, es posible que un mecanismo recomendado en una versión determinada de esta Guía, pase a ser considerado como heredado en versiones posteriores, si existen razones de peso para ello, como por ejemplo, por compatibilidad con implementaciones o arquitecturas previas. Esta cuestión es de especial relevancia si, por ejemplo, los avances de la computación cuántica son más rápidos de los que se consideran en la literatura.

16. De hecho, es sabido que los dos problemas computacionalmente difíciles en los que se fundamenta la seguridad de la mayoría de los criptosistemas asimétricos (o de clave pública), como son el problema de la factorización de enteros o IFP (*Integer Factorization Problem*) y el problema del logaritmo discreto o DLP (*Discrete*

Logarithm Problem) son vulnerables si se desarrolla un ordenador cuántico con la suficiente potencia de cómputo, dicho de otro modo, Shor publicó sendos algoritmos cuánticos que rompen ambos problemas en tiempo polinómico [Sho97]. De modo análogo, es conocido que si se dispusiera de un ordenador cuántico con la suficiente capacidad de cómputo, los criptosistemas de clave simétrica (o de clave secreta) verían restringida su seguridad a la mitad de las longitudes de las claves actuales, dado que el algoritmo propuesto por Grover reduce el tiempo de búsqueda por un ataque de fuerza bruta a la raíz cuadrada del tiempo actual [Gro97]. Además, resultados posteriores al de Grover, basados en la paralelización del algoritmo de Simon, apuntan a que esta seguridad podría reducirse aún más [KLLNP16a], [KLLNP16b].

17. La situación que acaba de comentarse sobre la vulnerabilidad cuántica de los principales mecanismos criptográficos actuales pone sobre la mesa uno de los problemas a los que se enfrenta la protección de la información almacenada en la actualidad. Se trata de tomar medidas de modo que esta información no pueda ser desvelada en el futuro si la computación cuántica (u otros ataques) es capaz de vulnerar los sistemas empleados en la actualidad y que la están protegiendo.

18. Así, por ejemplo, si la criptografía simétrica puede ver reducida su seguridad a la mitad de las longitudes de las claves usadas hoy en día, debería recomendarse el uso de sistemas cuya seguridad equivalente sea considerada segura, esto es, al menos, 256 bits (ya hemos mencionado que los mecanismos recomendados son aquellos que ofrecen una seguridad probada de, al menos, 128 bits).

19. Por su parte, para la criptografía asimétrica debería recomendarse el uso de soluciones híbridas de modo que la combinación de un mecanismo criptográfico determinado con un mecanismo criptográfico que se considere resistente a la computación cuántica, garantizaría por un lado, su seguridad frente a los ataques precuánticos (actuales), a la vez que proporcionaría determinada seguridad frente a los ataques cuánticos.

20. [**Protección cuántica:**] para los criptosistemas simétricos debería recomendarse el uso de claves que proporcionen una seguridad de, al menos, 256 bits; mientras que para la criptografía asimétrica debería recomendarse el uso de soluciones híbridas.

1.2. Mecanismos Criptográficos

21. Con el fin de establecer una terminología clara a lo largo de esta Guía, a continuación presentamos los principales conceptos utilizados. Es importante tener en cuenta que las definiciones que se ofrecen a continuación, como por ejemplo, la distinción entre esquemas criptográficos, protocolos criptográficos o primitivas de construcciones,

pueden considerarse en cierto modo arbitrarias, ya que los límites entre los diferentes conceptos pueden ser difusos.

1.2.1. Algoritmos, Protocolos y Esquemas

22. Con el fin de tener una referencia externa, seguimos la propuesta de [Sch21], que está en consonancia con las prácticas comunes de la comunidad criptográfica y parece ser compartida por otros organismos internacionales como [ACM1.3].

    - Un **Algoritmo Criptográfico** es una transformación, esto es, un conjunto de pasos, bien definida de modo que dado un valor de entrada, produce un valor de salida, logrando ciertos objetivos de seguridad.
    - Un **Protocolo Criptográfico** es un algoritmo distribuido que describe con precisión las interacciones entre dos o más entidades, logrando ciertos objetivos de seguridad.

      Al igual que con los algoritmos, se espera que un protocolo satisfaga ciertos objetivos de seguridad. Los protocolos criptográficos requieren la interacción entre dos o más partes y, por lo tanto, la definición de estos protocolos requiere la definición de los tipos de canales que están disponibles para estos usuarios.
    - Un **Esquema Criptográfico** es un conjunto de algoritmos criptográficos y protocolos criptográficos relacionados que logran ciertos objetivos de seguridad.

23. [**Algoritmo Criptográfico:**] conjunto de pasos que para una entrada dada proporciona una salida verificando determinados objetivos de seguridad.

24. [**Protocolo Criptográfico:**] algoritmo distribuido que describe las interacciones entre dos o más entidades, logrando ciertos objetivos de seguridad.

25. [**Esquema Criptográfico:**] conjunto de algoritmos y protocolos criptográficos relacionados que cumplen ciertos objetivos de seguridad.

26. En general, los mecanismos criptográficos se distinguen según su complejidad. Así, se denominan

    - **Primitiva Criptográfica:** es el mecanismo criptográfico más elemental.
    - **Construcción Criptográfica:** es un mecanismo criptográfico que permite lograr objetivos de seguridad de mayor nivel que las primitivas.

- **Primitiva Atómica:** es un mecanismo criptográfico que no puede describirse como una construcción o cuando tal descomposición no es una práctica habitual en la industria criptográfica.

> Nota 2 **[Primitivas y Construcciones autorizadas]** Una construcción se considera ≪autorizada≫ solo si utiliza primitivas autorizadas, a menos que se indique explícitamente lo contrario. Por otra parte, para que una construcción señalada como ≪recomendada≫ dé como resultado un mecanismo recomendado, es necesario que las primitivas subyacentes también sean recomendadas. De hecho, si una construcción está considerada como ≪recomendada≫, pero una primitiva subyacente está marcada como ≪heredada≫, el mecanismo resultante se considerará ≪heredado≫, no recomendado.

27.

28. Debe tenerse en cuenta que es posible que un mecanismo criptográfico dado sea simultáneamente una primitiva y una construcción, dependiendo de si forma parte como primitiva de un protocolo o de si está compuesto por otras primitivas criptográficas. A modo de ejemplo, el algoritmo de firma digital basado en curvas elípticas o ECDSA (*Elliptic Curve Digital Signature Algorithm*) puede considerarse como una construcción, dado que se basa en una función resumen (*hash*) criptográfica y en el logaritmo discreto en un grupo de puntos de una curva elíptica, y, a la vez, puede usarse como una primitiva en un protocolo de intercambio de claves autenticado.

29. Con relación a la dificultad mencionada de esos ≪límites difusos≫ en las definiciones anteriores, es importante señalar que, en el diseño de mecanismos criptográficos, la tendencia generalizada consiste en favorecer diseños modulares. Por ejemplo, la función resumen SHA-3 es una función de esponja construida sobre una permutación con buenas propiedades criptográficas; así, podría considerarse esta función tanto una primitiva como un esquema basado en esta permutación primitiva. También, por ejemplo, un esquema de cifrado implica una comunicación entre dos partes: una que cifra el texto claro y otra que descifra el texto cifrado, por lo que en lugar de ser un esquema podría considerarse como un protocolo elemental donde hay una interacción limitada entre las dos partes.

30. Ejemplos de primitivas atómicas son, por ejemplo, el estándar de cifrado avanzado o AES (*Advanced Encryption Standard*), la función resumen SHA-256, o problemas como el de la factorización de enteros y el del logaritmo discreto sobre cuerpos finitos (ya sea en su versión multiplicativa o aditiva).

31. En este sentido, es importante señalar que la fortaleza o robustez criptográfica de una primitiva atómica es difícil de evaluar dado que, en general, se debe evaluar la dificultad computacional de un problema matemático específico. Además, es muy probable que no se pueda dar una demostración matemática efectiva de tal

seguridad. De hecho, la seguridad de estos mecanismos se considera supuesta la dificultad computacional de determinado problema, esto es, de que no existe prueba evidente de que el problema puede resolverse en tiempo polinómico. Así, esta seguridad recae, en parte, en la confianza que el evaluador deposita en los resultados académicos publicados.

32. Dado que las entidades que interactúan en un protocolo criptográfico se intercambian información a través de canales de comunicación, es posible hacer una distinción básica entre los canales punto a punto que unen dos entidades y los canales de transmisión que conectan un remitente a varios receptores.

33. Muchas veces se da por hecho que los canales de comunicación ofrecen garantías de seguridad específicas. Así, suele entenderse que un «canal privado» (o canal seguro) es un canal punto a punto que utiliza alguna forma de cifrado para proteger la información intercambiada contra posibles escuchas y, además, es muy probable que emplee alguna forma de autenticación para evitar la manipulación de los mensajes. Por otra parte, un canal sin protección contra escuchas ilegales y manipulaciones suele denominarse «canal público» (o canal inseguro).

1.2.2. Tipos de Ataque

34. Los «modelos de comunicación» describen los tipos de canales que están disponibles entre diferentes conjuntos de entidades.

35. Por otra parte, también se distinguen los ataques pasivos de los activos, de modo que ambos tipos de ataque se definen en función del adversario o atacante. Si una entidad ha caído bajo el control de un adversario se dice que es «corrupta»; mientras que las entidades restantes se consideran «honestas».

    - En un **Ataque Pasivo**, el adversario no interfiere con la ejecución de los algoritmos y protocolos que componen un esquema criptográfico. Un adversario pasivo simplemente escucha la comunicación entre las entidades y registra toda la información a la que tiene acceso, incluida toda la información privada de las entidades corruptas.

      Las entidades que son pasivamente corruptas también se conocen como entidades semihonestas u «honestas pero curiosas».

    - En un **Ataque Activo**, el adversario, además de actuar pasivamente, puede interferir con la comunicación del esquema criptográfico borrando, inyectando o modificando mensajes. El adversario activo puede hacer que las entidades corruptas se desvíen de su comportamiento prescrito de manera arbitraria.

      Las entidades activamente corruptas suelen denominarse entidades «maliciosas» o tramposas.

36. [**Ataque Pasivo:**] ataque en el que el adversario no interfiere, solo escucha y registra, la comunicación entre las entidades.

37. [**Ataque Activo:**] ataque donde el adversario, además, interfiere en la comunicación del esquema criptográfico borrando, inyectando o modificando mensajes.

38. A modo de ejemplo, un ataque pasivo a un esquema de cifrado podría corresponderse con el comportamiento clásico de un fisgón; mientras que el conocido ataque del hombre en el medio o MitM (*Man-in-the-Middle*) a un protocolo de intercambio de claves es un caso de ataque activo.

1.2.3. Esquemas Simétricos y Asimétricos

39. Finalmente, haremos mención a los conceptos de criptografía simétrica y asimétrica. Generalmente los mecanismos criptográficos hacen uso de claves criptográficas. Su seguridad se fundamenta no solo en mantener la confidencialidad e integridad de las claves, sino también en cómo las mismas son empleadas en los diferentes esquemas criptográficos y sus correspondientes operaciones. Por ello es tradicional considerar los siguientes dos tipos de esquemas criptográficos:

    - **Esquema criptográfico simétrico:** es un esquema en el que la clave utilizada por cada parte es conocida o puede ser derivada por la otra parte. Las claves empleadas son únicas e idénticas para todas las partes que intervienen. En los esquemas simétricos, los intervinientes suelen llevar a cabo procesos análogos y comparten los parámetros a utilizar, de ahí el calificativo de ≪simétrico≫.

    - **Esquema criptográfico asimétrico:** es aquel esquema que, a diferencia de los esquemas simétricos, los parámetros y claves utilizados y los procesos realizados por cada parte son diferentes.

40. Los esquemas criptográficos simétricos y asimétricos siguen diseños muy diferentes y, por extensión, los mecanismos criptográficos se clasifican de forma análoga a los esquemas. Es interesante indicar que las funciones resumen son mecanismos criptográficos que no emplean claves y se consideran primitivas criptográficas simétricas.

1.3. Complejidad de un Ataque y Niveles de Seguridad

41. Es costumbre definir el «nivel de seguridad» de un mecanismo criptográfico como el número de operaciones necesarias para que un adversario rompa con éxito la

seguridad proporcionada por dicho mecanismo. mente este número se expresa como un logaritmo en base 2, de modo que una seguridad de 100 bits significa que son necesarias 2^{100} operaciones para romper el mecanismo.

1.3.1. Complejidad de un Ataque

42. Las diferentes métricas de complejidad que evalúan el coste de ejecutar un ataque son las siguientes:

    - La **Complejidad temporal** hace referencia a la cantidad de cálculos fuera de línea (*offline*) requeridos por el ataque, basados en datos extraídos del mecanismo criptográfico. A veces a esta complejidad se la conoce como «complejidad fuera de línea».

      Debe tenerse en cuenta que la complejidad temporal no depende de la potencia computacional de que dispone el atacante, de modo que la misma se establece sin considerar el modelo de computación que use el atacante y así es consistente con las mejoras que puedan producirse en la tecnología. Si el ataque es paralelizable, el atacante puede aprovechar esta característica para rebajar el tiempo que necesitaría si no lo fuera, pero esta propiedad no modifica la complejidad temporal porque no reduce la cantidad total de cálculos necesarios para llevar a cabo el ataque.

      Por ejemplo, la complejidad temporal de una búsqueda exhaustiva de claves AES-128 es (aproximadamente) 2^{128} operaciones.

    - La **Complejidad en memoria** considera la cantidad de almacenamiento que es necesario para realizar el ataque.

      Por ejemplo, la implementación de Nohl y Krißler [NK09] de la compensación de tiempo/memoria de Barkan, Biham y Keller [BBK03] en el algoritmo A5/1 requiere 2 TB de memoria para almacenar las tablas del arco iris.

    - La **Complejidad en datos** tiene en cuenta la cantidad de interacciones que el adversario necesita realizar con el mecanismo criptográfico para desarrollar el ataque contra él. Estas interacciones pueden corresponder a la cantidad de datos que el adversario necesita extraer de la ejecución de los mecanismos criptográficos para poder realizar el ataque, o a la cantidad de datos que el adversario debe enviar para lograr un resultado determinado. Esta complejidad suele calificarse como «complejidad en línea».

      A modo de ejemplo, el criptoanálisis lineal contra el cifrado de datos estándar o DES (*Data Encryption Standard*) de Matsui [Mat94] requiere 2^{43} textos claros conocidos, y el ataque de oráculo PKCS#1 v1.5 (*Public-Key Cryptography Standard*) de Bleichenbacher [Ble98] al criptosistema RSA propuesto por Rivest, Shamir y Adleman [RSA78] de 1024 bits necesita alrededor de un millón de consultas al oráculo de relleno.

43. [**Complejidad temporal:**] es la cantidad de cálculos fuera de línea necesarios para realizar con éxito un ataque criptográfico.

44. [**Complejidad en memoria:**] es la cantidad de almacenamiento necesario para ejecutar exitosamente un ataque.

45. [**Complejidad en datos:**] es la cantidad de interacciones que el adversario necesita realizar con el mecanismo criptográfico para desarrollar el ataque.

46. Todas estas medidas de complejidad también se pueden combinar compensando unas con otras. Así, es posible disminuir la complejidad temporal de un ataque aumentando la complejidad en memoria o en datos.

1.3.2. Niveles de Seguridad

47. Al hablar del nivel de seguridad de un mecanismo criptográfico, en general, se entiende que se está considerando como la complejidad temporal del mejor ataque conocido, de modo que las complejidades en memoria y en datos pasan a un segundo término. De hecho, el coste de las operaciones necesarias para la escritura de la información en la memoria y el procesamiento de los datos se supone que ya se han tenido en cuenta cuando se determinó la complejidad temporal de determinado ataque. Esto es, el nivel de seguridad de un mecanismo no puede ser menor que la complejidad en memoria o la complejidad en datos del mejor ataque conocido al mecanismo.

48. [**Nivel de seguridad de un mecanismo criptográfico:**] es la complejidad temporal del mejor ataque conocido contra dicho mecanismo. La complejidad en memoria y en datos pasan a un segundo término.

49. Al diseñar o evaluar un mecanismo criptográfico, debe llevarse a cabo un análisis sobre la fortaleza del mismo, de modo que debe haber ciertas garantías de que ningún atacante pueda romper la seguridad del mecanismo. Por ello, al proponer el uso de un mecanismos dado, se deben recomendar, a la vez, determinados parámetros, en función de la seguridad deseada. Así, es importante recomendar, por ejemplo, el tamaño de la clave y el tamaño del estado interno.

50. Algunos principios que deben considerarse en el proceso de evaluación de los mecanismos criptográficos son los siguientes:

    - Como ya se ha mencionado anteriormente, los mecanismos recomendados deben proporcionar una seguridad de, al menos, 125 bits contra ataques temporales (fuera de línea). Una seguridad de 100 bits solo es aceptable para

los mecanismos heredados, que claramente proporcionan una seguridad más baja.

- En situaciones muy especiales, se podría aceptar una complejidad temporal menor. Podría considerarse esta situación cuando se utilizan claves de sesión, esto es, claves de un solo uso, o en los mecanismos criptográficos denominados ligeros, como los implementados en etiquetas de identificación por radio frecuencia o RFID (*Radio Frequency Identification*), cuyos recursos computacionales son limitados.

- Los niveles de seguridad aceptables asociados a la complejidad en datos suelen ser menores. Esto se debe a que adquirir o transmitir datos a un proceso, dispositivo, etc., que utiliza alguna clave secreta precisa comunicarse con dicho dispositivo. En este caso, hay restricciones en la comunicación que vienen determinadas por las limitaciones físicas del dispositivo (tarjeta inteligente, etc.), o que pueden ser impuestas por el mismo (número máximo de intentos de identificación mediante un PIN, tiempo de caducidad de las claves, etc.). Todo ello indica que basta con estudiar la seguridad del mecanismo contra los atacantes cuya complejidad en datos está limitada o restringida.

  Por ejemplo, dado que en una red a 100 Gb/s, el tiempo necesario para intercambiar 2^{64} bloques de AES es aproximadamente setecientos años, los posibles ataques contra AES que requieran acceder a más de 2^{64} bloques de datos no son un problema real.

- Es muy importante tener en cuenta el estado del arte relacionado con la seguridad de los mecanismos criptográficos. Es posible que cuando un mecanismo fuera propuesto y aceptado (incluso considerado como un estándar) para su uso con determinados valores de sus parámetros, sea criptoanalizado con posterioridad. Es decir, cabe en lo posible que investigaciones posteriores hayan determinado que el mecanismo no es seguro para determinado conjunto de parámetros. En este caso, la existencia de posibles ataques invalidan la propuesta original, incluso en el caso de que estos nuevos ataques no tengan un impacto inmediato práctico.

51. Ya hemos mencionado que el concepto de nivel de seguridad de un mecanismo se expresa en términos de la cantidad mínima de cálculos que deben realizarse para romperlo y que ello se traduce en términos del tamaño de las claves. Conviene, por tanto, señalar que cuando mencionamos la noción de «longitud o tamaño de una clave», nos estamos refiriendo a la entropía del mecanismo que genera dicha clave.

52. A modo de ejemplo, es sabido que las claves de DES se almacenan en 64 bits y que 8 de esos bits (el último de cada byte) son redundantes, esto es, son los bits de paridad que verifican que cada grupo de 7 bits que precede a cada uno de ellos es correcto. Así pues, la longitud de la clave de DES es, realmente, de solo 56 bits. Como consecuencia, la longitud de la clave de Triple-DES con 2 claves (resp. 3

claves) es de 112 bits (resp. 168 bits), que es estrictamente menor que los 128 bits (resp. 192 bits) utilizados para su almacenamiento.

53. En el caso de la claves correspondientes a los mecanismos asimétricos, la relación entre la longitud de la clave y el nivel de seguridad no es tan directa y tan sencilla de expresar. De hecho, en general, la seguridad de una clave asimétrica depende de un problema matemático, supuestamente difícil de resolver, y de la complejidad temporal del mejor algoritmo conocido que sea capaz de resolver dicho problema.

54. A modo de ejemplo, si la clave, n, de un mecanismo criptográfico asimétrico es producto de dos números primos con, aproximadamente, el mismo tamaño, y tiene una longitud de $b = 1024$ bits (resp. $b = 2048$ bits) y el mejor algoritmo conocido para resolver el problema en el que se basa la seguridad de dicho mecanismo require $\mathcal{O}\left(e^{(1,92\cdot b)^{1/3}\cdot(\log_2(b))^{2/3}}\right)$ operaciones[2], se deduce que la seguridad proporcionada por esta clave es de $2^{83,89}$ operaciones (resp. $2^{112,63}$ operaciones), que sería equivalente a una seguridad de unos 83 bits en el primer caso (resp. 112 bits).

1.4. Organización de la Guía

55. Después de este primer capítulo introductorio, el resto de esta Guía se organiza de la siguiente manera. En el capítulo 2 se presentan las primitivas atómicas y construcciones criptográficas simétricas. El capítulo 3 introduce las primitivas atómicas y construcciones criptográficas asimétricas. El capítulo 4 incluye uno de los principales protocolos criptográficos de comunicación como es el protocolo de seguridad de la capa de transporte o TLS (*Transport Layer Security*), además del protocolo SSH y del protocolo IPSec con IKEv2. La idea es ir complementado este capítulo con más protocolos en el futuro. Los generadores de números aleatorios se detallan en el capítulo 5; mientras que los protocolos de gestión de claves se contemplan en el capítulo 6 y en el capítulo 7 se incluyen los mecanismos de autenticación e identificación. En el anexo A se presenta la generación de números primos y de claves para el criptosistema asimétrico RSA. por su parte, en el anexo B se tratan los principales fundamentos en los que se basa la criptografía postcuántica, que propone soluciones criptográficas seguras para ser implementadas en ordenadores como los actuales, pero capaces de resistir la potencia de cómputo de los futuros ordenadores cuánticos. Finalmente, en el anexo C se comentan los esquemas de firma digital basados en el problema del logaritmo discreto.

56. Cada una de las primitivas y construcciones autorizadas estará acompañada de una tabla en la que se incluirán sus nombres o denominaciones, los tamaños de los parámetros (si ha lugar), si se considera recomendada o heredada (denotándose

[2]Se ha elegido esta función a modo de ejemplo y con fines didácticos, aunque está estrechamente relacionada con el problema de la factorización de enteros, que es la base de la seguridad del criptosistema RSA.

cada propiedad como R o H, respectivamente) y otros aspectos que puedan ser de interés.

57. Este documento incluye, además, un Glosario de términos para facilitar su lectura y una Bibliografía que permitirá al lector ampliar determinados aspectos introducidos en esta Guía, en la que no tiene cabida un desarrollo en profundidad de los mismos.

2. Mecanismos Simétricos

58. Como ya se ha mencionado en la Sección 1, la seguridad de la criptografía simétrica se basa, fundamentalmente, en mantener la confidencialidad de una clave que comparten los usuarios legítimos que intervienen en un proceso criptográfico dado. En esta sección trataremos los mecanismos que entran dentro del ámbito de los considerados simétricos.

2.1. Primitivas Simétricas

59. La criptografía simétrica (o de clave secreta) se caracteriza porque los intervinientes en un proceso de cifrado utilizan una única clave para cifrar y descifrar la información que se intercambian. Dicha clave es la misma para ambos procesos, por lo que debe ser compartida. La seguridad de estos mecanismos se basa, por lo tanto, en mantener en secreto tal clave compartida.

60. En esta sección presentamos las primitivas atómicas simétricas aprobadas, para describir posteriormente las construcciones simétricas construidas sobre estas primitivas.

2.1.1. Cifradores en Flujo

61. El procedimiento para cifrar en flujo un texto claro de L bits consiste en generar una secuencia de L bits, aparentemente aleatorios, a partir de una clave secreta corta de k bits, que es conocida solo por las dos partes interesadas), y un algoritmo público que genera la secuencia de los L bits. Esta secuencia generada se denomina secuencia de flujo de claves (*keystream sequence*).

62. Para el cifrado, el remitente realiza la operación XOR bit a bit entre los bits del texto claro y la secuencia de flujo de claves. El resultado es el texto cifrado que se enviará al receptor. Para el descifrado, el receptor genera la misma secuencia de flujo de claves, realiza la misma operación XOR bit a bit entre el texto cifrado recibido y la secuencia generada, recuperando el texto claro original. El objetivo es hacer prácticamente imposible para un adversario la recuperación del texto claro a partir del texto cifrado sin el conocimiento de la clave secreta k.

63. En la versión actual de esta Guía no se recomienda ningún cifrado en flujo concreto, por lo que no se presenta ninguna tabla de primitivas de cifrado.

64. Es sabido que en 2004 se lanzó el proyecto eSTREAM como parte de ECRYPT (European Network of Excellence in Cryptology[3]). El proyecto es un esfuerzo de varios años para identificar nuevos cifrados en flujo que podrían ser adecuados

[3] http://www.ecrypt.eu.org/stream/

para una adopción generalizada. Como resultado, en 2008 se preseleccionaron siete finalistas (cuatro de perfil software y tres de perfil hardware), todos ellos fueron declarados aptos porque todos ellos son algoritmos rápidos y adecuados para el cifrado en flujo, pero no se han convertido en estándares.

2.1.2. Cifradores en Bloque

65. El cifrado en bloque es un método que permite cifrar cada uno de los bloques en los que se divide un mensaje de texto claro, cada uno de ellos de la misma longitud, sea L bits, dando como resultado un bloque cifrado, también de L bits de longitud, la misma que el original. La operación de cifrado viene determinada por una clave secreta de r bits de longitud, elegida uniformemente al azar. La operación inversa, esto es, el descifrado de cada bloque, utiliza la misma clave que para el cifrado y devuelve el bloque del texto claro original. El objetivo de este tipo de cifrado es hacer que sea prácticamente imposible recuperar el texto claro a partir del texto cifrado si no se conoce la clave secreta utilizada.

66. Desde el punto de vista de su implementación, un cifrado en bloque puede entenderse como una familia de permutaciones del conjunto $\{0,1\}^n$, cada una de las cuales se selecciona haciendo uso de un parámetro de r bits llamado clave. Los valores n y r representan, respectivamente, el tamaño en bits del bloque y el tamaño de la clave.

67. En la Tabla 2.1 se listan las primitivas autorizadas, los tamaños de sus parámetros, si se considera Recomendada o Heredada y otros aspectos que puedan ser de interés.

Primitiva	Parámetros (bits)	Rec./Her.	Referencias	Notas
AES	$k = 128$	$R^{\varnothing}$	[FIPS197], [ISO18033-3]	3
	$k = 192$	R		
	$k = 256$	R		

Tabla 2.1: Tipos autorizados de cifradores en bloque

68. Nota 3 **[Amenaza de la Computación Cuántica]** Los ataques a los cifradores en bloque se pueden acelerar utilizando algoritmos que aprovechan la potencia de los ordenadores cuánticos (algoritmo de Grover). Estos ataques se pueden realizar retroactivamente, es decir, los mensajes cifrados se pueden almacenar ahora para ser descifrados más adelante cuando existan ordenadores cuánticos suficientemente potentes. En los escenarios donde sea necesario ofrecer resistencia a los ataques que aprovechen los ordenadores cuánticos no se recomienda utilizar cifradores en bloque con tamaño de clave menor a 192 bits. Los algoritmos que incumplen con este requisito se muestran con el símbolo $R^{\varnothing}$.

2.1.3. Funciones Resumen

69. Una función resumen (*hash* en inglés) es una función, sin clave, computacionalmente eficiente, $\mathfrak{h}$, que aplica cadenas binarias de longitud arbitraria en cadenas binarias de longitud fija [MvV96, Cap. 9]. La salida de este tipo de funciones se llama «resumen» o «valor hash». La idea principal que subyace a estas funciones es que su salida o resumen puede usarse como una representación compacta de una cadena de entrada de longitud arbitraria. Las funciones resumen se emplean, preferentemente, en esquemas de firma digital y para verificar la integridad de datos.

70. Las funciones resumen deben verificar las siguientes cuatro propiedades para ser consideradas seguras:

    1. «Resistencia a la preimagen»: para un resumen dado, y, es computacionalmente imposible encontrar una entrada, x, cuya salida sea precisamente y, es decir, es inviable computacionalmente encontrar una preimagen x tal que $\mathfrak{h}(x) = y$. En otras palabras, debe ser difícil invertir la función $\mathfrak{h}$.

    2. «Resistencia a la segunda preimagen»: dada una entrada x_1, es computacionalmente imposible encontrar una segunda entrada diferente, x_2, que tenga la misma salida, es decir, encontrar una segunda preimagen $x_2 \neq x_1$ tal que $\mathfrak{h}(x_1) = \mathfrak{h}(x_2)$.

    3. «Resistencia a colisiones»: no es factible, computacionalmente hablando, encontrar dos entradas distintas x_1 y x_2 que tengan la misma salida, es decir, que $\mathfrak{h}(x_1) = \mathfrak{h}(x_2)$.

    4. «Resistencia cercana a la colisión»: es difícil encontrar dos entradas distintas, x_1 y x_2 de modo que la diferencia entre $\mathfrak{h}(x_1)$ y $\mathfrak{h}(x_2)$ $(\mathfrak{h}(x_1) \oplus \mathfrak{h}(x_2))$ sea pequeña.

71. Es sabido que las funciones *hash* de la familias SHA2 y SHA3 verifican estas condiciones, por lo que se consideran funciones autorizadas y de ahí su inclusión en esta guía.

72. En la Tabla 2.2 se listan las funciones resumen autorizadas, los tamaños de sus resúmenes (h) y la denominación de la función resumen correspondiente, si se considera Recomendada o Heredada y con qué plazo de caducidad y las referencias apropiadas.

Primitiva	Parámetros (bits)	Rec./Her.	Referencias	Notas
SHA-2	$h = 256$ (SHA-256) $h = 384$ (SHA-384) $h = 512$ (SHA-512) $h = 256$ (SHA-512/256)	$R^{\varnothing}$ R R $R^{\varnothing}$	[FIPS180-4], [ISO10118-3]	4
SHA-3	$h = 256$ $h = 384$ $h = 512$	$R^{\varnothing}$ R R	[FIPS202]	4

Tabla 2.2: Funciones resumen autorizadas

73. Nota 4 **[Amenaza de la Computación Cuántica]** Existen algoritmos que aprovechan los ordenadores cuánticos y que pueden acelerar los ataques contra las funciones hash. En contextos donde se requiere resistencia contra ataques que utilizan ordenadores cuánticos, se recomienda no usar las funciones hash con $h = 256$ bits. Los algoritmos que incumplen con este requisito se muestran con el símbolo $R^{\varnothing}$.

74. La función SHA-1 no es una función resumen autorizada; sin embargo, el código de autenticación de mensajes conocido como HMAC-SHA-1, cuya construcción se basa en SHA-1, se acepta como un esquema heredado, como se verá en la sección 2.2.3.

2.1.4. Funciones resumen con salida variable

75. Las funciones de salida extensible o variable (*Extendable-Output Function*, XOF) son una extensión de las funciones hash criptográficas que pueden generar salidas de longitud arbitraria, lo cual representa una desviación significativa de las funciones hash de salida de longitud fija. Encuentran su utilidad en diversas aplicaciones criptográficas donde se desean valores hash variables o muy largos.

76. En particular, a partir de la familia SHA-3 se puede definir una familia de funciones con salidas extensibles, esto es, funciones como la SHA-3 pero con una salida infinita que puede ser fijada. La familia se conoce como SHAKE (*Secure Hash Algorithm and Keccak*) y las dos versiones más utilizadas son SHAKE-128 y SHAKE-256, que proporcionan salidas de 128 y 256 bits, respectivamente [FIPS202]. Si m es un mensaje y d el número de bits de salida, la función SHAKE-XXX(d, m), con XXX=128 o 256, ofrece una seguridad contra colisiones de $\min(d/2, \text{XXX})$ y con respecto a la segunda preimagen de $\min(d, \text{XXX})$. Dado que el menor de estos valores ha de ser superior a 256 bits, solo cabe considerar la función SHAKE-256 con $d = 512$, de modo que la seguridad contra colisiones es $\min(512/2, 256) = 256$, y contra la segunda preimagen es $\min(512, 256) = 256$.

77. SHAKE se basa en la función de permutación de Keccak y actúa de forma muy parecida a como lo hace SHA-3, si bien con dos diferencias. La primera es que se usa un separador de dominio diferente (se agrega 1111 a la entrada en lugar de 01 para SHA-3) y la segunda es que la función esponja de Keccak se puede seguir apretando tanto como se quiera, mientras que SHA-3 define cuánto se debe apretar.

78. La función cSHAKE [SP800-185] es una versión personalizable de SHAKE por lo que funciona igual que esta, a excepción de que tiene una nueva entrada, que es una cadena de personalización. Su propósito es el de evitar colisiones entre diferentes usos de la función SHAKE. Existen dos variantes de cSHAKE: cSHAKE-128 y cSHAKE-256, que ofrecen una seguridad de 128 y 256 bits, respectivamente. Por este motivo, solo la función cSHAKE-256 está autorizada.

Primitiva	Fortaleza de seguridad	Rec./Her.	Referencias	Notas
SHAKE	$s = 128$ (SHAKE128) $s = 256$ (SHAKE256)	$R^{\varnothing}$ R	[FIPS202]	5, 6
cSHAKE	$s = 128$ (cSHAKE128) $s = 256$ (cSHAKE256)	$R^{\varnothing}$ R	[SP800-185]	5, 6

Tabla 2.3: Primitivas XOF autorizadas

79. Nota 5 [**Funciones XOF**] Las primitivas criptográficas XOF deben ser siempre implementadas como primitivas subyacentes de construcciones criptográficas como KMAC o XMSS. Por lo tanto, su uso independiente se considera no recomendado.

80. Nota 6 [**Amenaza de la Computación Cuántica**] Existen algoritmos que aprovechan los ordenadores cuánticos y que pueden acelerar los ataques contra las funciones hash y las XOFs. En contextos donde se requiere resistencia contra ataques que utilizan ordenadores cuánticos, se recomienda no usar las funciones XOF con $s = 128$. Los algoritmos que incumplen con este requisito se muestran con el símbolo $R^{\varnothing}$.

2.1.5. Compartición de Secretos

81. Los esquemas de intercambio, división o compartición de secretos son métodos que distribuyen un secreto entre un grupo de participantes, de modo que cada participante recibe una parte o sombra del secreto. La recuperación del secreto

original solo puede llevarse a cabo cuando determinado número de participantes comparten su sombra con el resto.

82. De forma más precisa, se denomina esquema de compartición de secretos (k, n)-umbral a un protocolo que permite repartir un secreto en varias sombras, una para cada participante, de modo que el conocimiento de, al menos, k sombras permite recuperar el secreto, pero es imposible hacerlo con $k-1$ sombras o menos [Sha79]. Debe tenerse en cuenta que el conocimiento de cualquier sombra no aporta ninguna información sobre el secreto original.

83. La Tabla 2.4 muestra el único esquema de compartición de secretos autorizado, que corresponde a la propuesta de Shamir del año 1979.

Primitiva	Rec./Her.	Referencias	Notas
Compartición de secretos de Shamir	R	[Sha79]	

Tabla 2.4: Compartición de secretos autorizada

2.2. Construcciones Simétricas

84. Las construcciones simétricas se basan en las primitivas simétricas ya presentadas y permiten tratar una gran variedad de objetivos de seguridad. La seguridad de los esquemas simétricos con clave (secreta) se fundamenta, al igual que las primitivas en las que se basan, en la confidencialidad e integridad de una clave secreta compartida por las partes legítimas que intervienen en el proceso o comunicación. En el capítulo 6 se tratarán los mecanismos para la generación de claves secretas y determinados aspectos sobre su gestión.

2.2.1. Modos de operación en el Cifrado y Descifrado

85. Cada cifrador en bloque utiliza diferentes «modos de operación», los cuales permiten gestionar de diferente manera, para los procesos de cifrado y descifrado, los bloques en los que se divide el texto claro o cifrado. El objetivo que persiguen estos modos de operación es proporcionar diferentes objetivos de seguridad a la hora de cifrar o descifrar un texto. Cada modo describe cómo aplicar de forma iterada una operación de cifrado (descifrado) en un bloque simple para transformar de modo seguro textos claros (cifrados) mayores que un único bloque.

86. La mayoría de los modos de operación utiliza una secuencia binaria única, llamada vector de inicialización o IV (*Initialization Vector*), para cada operación de cifrado. Este IV se utiliza para garantizar que se generan textos cifrados distintos aunque el mismo bloque se cifre varias veces con la misma clave. En algunos modos, el IV se puede elegir aleatoriamente.

87. En la Tabla 2.5 se muestran los diferentes modos de esquemas de cifrado simétrico con sus principales características.

Modo de operación	Rec./Her.	Referencias	Notas
CTR	R*	[SP800-38A], [ISO10116]	7, 8, 9
OFB	R*	[SP800-38A], [ISO10116]	7, 8, 9
CBC	R*	[SP800-38A], [ISO10116]	7, 8, 10
CBC-CS	R*	[SP800-38A-Add]	7, 8
CFB	R*	[SP800-38A], [ISO10116]	7, 8, 10

Tabla 2.5: Modos autorizados de cifrado simétrico

88. Nota 7 **[Tipo de Vector de Inicialización]** Con el fin de proporcionar una seguridad en un sentido estricto, el esquema de cifrado considerado debe ser probabilístico y generar un IV aleatorio para iniciar el proceso cifrado, o utilizar una entrada adicional, como un *nonce*. Cada una de las especificaciones de los diferentes modos de operación describen qué se utiliza, si un IV aleatorio o un *nonce*. Las implementaciones de estos modos deberán seguir estas especificaciones.

89. Nota 8 **[Integridad añadida]** Es sabido que los esquemas de cifrado autenticado ofrecen garantías de seguridad de privacidad superiores a las que brindan los mecanismos que solo cifran. Como consecuencia, aunque todos los modos que solo cifran, considerados anteriormente, están recomendados para la construcción de modos, su uso independiente se considera heredado. Esta es la razón de que aparezca la expresión R* en la Tabla 2.5.

90. Nota 9 **[Modo en flujo]** Algunos esquemas simétricos operan enmascarando el texto claro mediante un flujo de claves generado a partir de la propia clave y del IV. Estos modos de funcionamiento se llaman en flujo y para ellos es muy importante verificar que nunca se superponen dos flujos de claves. Esta propiedad se puede lograr de forma determinista en algunos modos, por ejemplo, en el CTR. En otros modos, como el OFB, la probabilidad de que tal propiedad se cumpla es casi 1, si no se utiliza el mismo par IV-clave (o solo con una probabilidad insignificante) para cifrar dos mensajes con la misma clave.

Nota 10 [***Padding* (Relleno)**] Algunos modos de cifrado no son capaces de tratar el cifrado del último bloque si este está incompleto, es decir, si tiene menos bits que los obligados para cada bloque. En estos casos, se debe llevar a cabo una modificación de tal bloque para que la operación pueda ejecutarse con normalidad; operación que suele denominarse padding. Uno de los procedimientos más utilizados consiste en completar el bloque claro con algunos datos estructurados de modo que se obtenga el tamaño requerido o que lo conviertan en un texto claro con una longitud que sea un múltiplo del tamaño del bloque. Sin embargo, la verificación del formato del padding durante el descifrado puede filtrar información sobre el valor descifrado de forma que podría usarse para descifrar cualquier bloque cifrado. Los desarrolladores y evaluadores deben asegurarse de que la implementación del descifrado no proporcione al atacante ningún tipo de información sobre el padding o su formato. Una alternativa a la aplicación del esquema de padding es el uso del robo de texto cifrado.

91.

2.2.2. Cifrado de Disco Duro

92. Como ya hemos mencionado en §2.2.1, los modos de cifrado de propósito general permiten cifrar y descifrar datos atendiendo a diferentes características y según diferentes niveles de seguridad. Además, para lograr una seguridad en un sentido estricto, en determinadas ocasiones hace falta expandir los datos de algunos bloques debido al uso de un vector de inicialización o *nonce*. Sin embargo, en algunos entornos, estas modificaciones resultan ser un inconveniente. Este es el caso, por ejemplo, del cifrado del disco duro. En estas situaciones se permite el uso de modos de cifrado deterministas, como se muestra a continuación, si bien deben tenerse en cuenta las notas que se incluyen en cada caso.

93. El modo de cifrado en bloque modificable XEX[4] con robo de texto cifrado o XTS (*XEX Tweakable Block Cipher with Ciphertext Stealing*) es un modo de uso de confidencialidad de cifradores en bloque que se utiliza para el cifrado de medios de almacenamiento de datos, sin aportar autenticación. Este modo está especialmente diseñado para cifrar los datos de un disco duro en el que el índice del sector del disco duro forma parte del proceso de cifrado. De esta manera la información queda protegida por dos parámetros: su ubicación y la clave utilizada, de modo que si un adversario cambia de disco o el sector donde se almacena la información, no podrá recuperarla aunque conozca la clave.

94. En la Tabla 2.6 se muestran los diferentes modos de esquemas de cifrado autorizados para cifrado de disco, con sus principales características.

[4] XEX significa XOR *Encrypt* XOR

Esquema	Rec./Her.	Referencias	Notas
XTS	R	[SP800-38E]	11, 12, 13, 14

Tabla 2.6: Modos autorizados de cifrado simétrico para cifrado de disco

95. Nota 11 [**Modo de cifrado en flujo de disco**] Los modos de cifrado de disco son, por naturaleza, modos de cifrado deterministas, donde el IV se deriva de la ubicación donde se almacenan los datos cifrados. Por ello, los modos de operación de cifrado en flujo son inapropiados, dado que los textos cifrados correspondientes a dos textos claros diferentes almacenados en la misma ubicación filtrarían la diferencia entre los textos en claro.

96. Nota 12 [**Tweak único**] El valor tweak que se utilice para cifrar cada posición de bloque (completo o incompleto) en cada sector de disco deber ser único, es decir, un disco o conjunto de discos cifrados con la misma clave nunca deberá contener dos bloques distintos cifrados con la misma clave y el mismo valor tweak.

97. Nota 13 [**Tweak de dirección**] Por lo general, el tweak se deriva de la dirección donde se almacena el texto cifrado. Para los dispositivos de almacenamiento o controladores de disco que determinan el tweak utilizado a partir de direcciones lógicas en lugar de direcciones físicas, la unicidad del tweak no está garantizada a priori y se debe tener especial cuidado para evaluar la conformidad con la Nota 12.

98. Nota 14 [**Claves en XTS-AES**] Es conveniente asegurarse que en el modo XTS-AES las claves K_1 y K_2 son diferentes [FIPS-140IG].

2.2.3. Códigos de Autenticación de Mensajes

99. Los códigos de autenticación de mensajes o MAC (*Message Authentication Codes*) ofrecen integridad y autenticación del origen de datos basada en un bloque de bits generado mediante un mecanismo simétrico, en general un cifrado en bloque o una

función resumen. Un MAC consta de dos funciones, una que genera el código y otra que lo verifica. La primera tiene como entradas una clave secreta y un mensaje y proporciona como salida el código. La segunda tiene como entrada la misma clave secreta, el mismo mensaje y el código, siendo su salida un elemento del conjunto $\{Verdadero, Falso\}$.

100. Para ahorrar ancho de banda en las comunicaciones, es costumbre truncar el resultado de un esquema MAC, pero a la vez, para que el esquema sea resistente a los ataques de suposición (*guessing*), en los que un adversario intenta falsificar el MAC mediante un valor aleatorio, la longitud final del MAC no debe ser demasiado corta.

101. El MAC basado en funciones resumen se denota por HMAC. La seguridad de este HMAC depende directamente de la seguridad de la función resumen que se utilice y se recomienda utilizar una función con al menos 256 bits de seguridad. A pesar de que la función resumen SHA-1 no está autorizada se acepta su uso con HMAC hasta el año 2030.

102. El algoritmo KMAC (*Keccak Message Authentication Code*) consta de una función pseudo-aleatoria y una función hash con clave basada en Keccak. KMAC proporciona una salida de longitud variable y la modificación de la longitud de salida genera una nueva salida no relacionada. Por este motivo en vez de truncar la salida de KMAC se recomienda utilizar el parámetro definido para tal efecto. KMAC tiene dos variantes: KMAC-128 y KMAC-256.

103. En la Tabla 2.7 se incluyen los códigos de autenticación de mensajes basados en cifradores en bloque y en funciones resumen que están autorizados.

Esquema	Tamaño clave (bits)	Rec./Her.	Referencias	Notas
CMAC		R	[SP800-38B], [ISO9797-1]	15, 16
CBC-MAC		R	[ISO9797-1, Alg. 1, Padd. 2]	15, 16, 17
HMAC	$k \geq 128$ $k \geq 100$	$R^{\varnothing}$ H	[RFC2104], [ISO9797-2]	15, 16, 23
HMAC-SHA1	$k \geq 100$	H[2030]	[RFC2104], [ISO9797-2], [FIPS180-4]	15, 16, 18, 23
GMAC		R	[SP800-38D]	15, 16, 19, 20, 21
KMAC-128	$k \geq 128$	$R^{\varnothing}$	[SP800-185]	22, 23
KMAC-256	$k \geq 256$	R	[SP800-185]	22

Tabla 2.7: MAC autorizados basados en cifradores en bloque y funciones resumen

104. Nota 15 [**MAC truncado a 96 bits**] Se acepta el truncamiento de un MAC a, al menos, 96 bits, generado por un mecanismo MAC autorizado. Esta condición necesaria no tiene por qué ser una condición suficiente para determinados esquemas MAC, como el GMAC (ver Nota 21).

105. Nota 16 [**MAC truncado a 64 bits**] El truncamiento de un MAC a, al menos, 64 bits, generado por un mecanismo de autorizado se considera heredado siempre que el número máximo de verificaciones del MAC realizadas para una clave determinada durante su vida útil pueda limitarse a 2^{20}.

106. Nota 17 [**Longitud de entrada fija**] El CBC-MAC solo se emplea en entornos en los que los tamaños de todas las entradas para las que se calcula el MAC con la misma clave son los mismos. Si se permiten entradas de longitud variable, es posible llevar a cabo falsificaciones de extensión de la longitud.

107. Nota 18 [**HMAC-SHA-1**] HMAC no requiere la resistencia a colisiones de la función hash subyacente y, por ahora, HMAC-SHA-1 se considera un mecanismo heredado aceptable, aunque la función resumen SHA-1 no lo sea. En todo caso, se recomienda eliminar gradualmente el HMAC-SHA-1.

108. Nota 19 [**GMAC-GCM con *nonce***] El IV debe administrarse dentro de las condiciones de seguridad del proceso de cifrado autenticado. Por ejemplo, es crucial asegurarse de que ningún adversario pueda hacer que el mismo IV sea reutilizado para proteger diferentes pares de (texto claro, datos asociados) con la misma clave.

109. Nota 20 [**Opciones para GMAC-GCM**] Solo están autorizadas las siguientes opciones para GCM: la longitud del IV debe ser igual a 96 bits, se debe utilizar el método de construcción determinista para IV dado en [SP800-38D, §8.2.1], la longitud del MAC debe ser 128 bits.

110. Nota 21 [**Límites para GMAC-GCM**] Como los límites de imposibilidad de falsificación de las opciones autorizadas de GMAC y GCM de la Nota 20 no son óptimos, la longitud del MAC debe ser de, al menos, 128 bits. Por lo tanto, no se debe realizar ningún truncamiento a una longitud MAC final, como 96 bits.

111. Nota 22 [**Truncamiento de KMAC**] Dado que la función de KMAC tiene un parámetro que permite indicar la longitud del MAC, se recomienda utilizar tal parámetro de forma explícita en lugar de truncar el resultado del KMAC para obtener la longitud deseada. En cualquier caso se deben respectar las longitudes mínimas indicadas en las Notas 15 y 16.

112. Nota 23 [**Amenaza de la Computación Cuántica**] Existen algoritmos que aprovechan los oredenadores cuánticos, como el algoritmo de Grover, que pueden acelerar los ataques contra los esquemas MAC basados en funciones hash. En contextos donde se requiere resistencia contra ataques que utilizan ordenadores cuánticos, se recomienda no usar esquemas MAC con claves de tamaño menor a 192 bits. Los algoritmos que incumplen con este requisito se muestran con el símbolo $R^{\varnothing}$.

2.2.4. Esquemas Simétricos de Autenticación de Entidades

113. Los esquemas de autenticación de entidades permiten que una entidad pruebe su identidad ante un verificador demostrando que conoce determinado secreto. Por su estructura, son esquemas interactivos y, en general, utilizan un esquema MAC o un esquema de cifrado con un protocolo de desafío-respuesta aleatorio. Para este tipo de esquemas de autenticación no se proporciona ninguna tabla de esquemas autorizados en concreto, dado que son esquemas con diferentes objetivos de seguridad a los de los MAC aunque puedan basarse en ellos. Así, los modos de integridad y los esquemas de autenticación de entidad simétricos no deben utilizar la misma clave (véase la Nota 102). Una condición necesaria para que se acuerde un esquema basado en un esquema de cifrado (resp. MAC) es que el cifrado en bloque subyacente (resp. MAC) esté autorizado.

114. Sin embargo, en la Tabla 2.8 sí se muestran los tamaños de los protocolos de desafío-respuesta autorizados.

Tamaño protocolo desafío-respuesta (bits)	Rec./Her.	Notas
$125 \leq l$	R	24
$96 \leq l < 125$	H	24

Tabla 2.8: Tamaño de los protocolos de desafío-respuesta autorizados

115. Nota 24 **[Protocolo desafío-respuesta]** El verificador debe asegurarse de que un desafío no puede repetirse, con una probabilidad no despreciable. El desafío podría, por ejemplo, implementarse mediante un valor aleatorio generado por el verificador, cuyo tamaño sea lo suficientemente grande.

2.2.5. Cifrado Autenticado

116. Los esquemas de cifrado autenticado o AE (*Authenticated Encryption*) permiten obtener confidencialidad, integridad y autenticación del origen de los datos o de los mensajes. Un esquema AE consta de una función de cifrado que transforma un texto claro en un texto cifrado mediante una clave determinada, y una función de descifrado que recupera el texto claro a partir del texto cifrado y la clave. La diferencia con los esquemas que solo cifran es que la función de descifrado del esquema AE puede no devolver un valor descifrado si el texto cifrado no respeta algún patrón de redundancia. Debe tenerse en cuenta que, por lo general, alguna parte del texto cifrado actúa como un valor de verificación que se comprueba durante el descifrado.

117. En muchas ocasiones, los AE incorporan una característica adicional que consiste en combinar la autenticación de los datos cifrados con la autenticación de datos adicionales no cifrados. El AE con esa propiedad se conoce como cifrado autenticado con datos asociados o AEAD (*Authenticated Encryption with Associated Data*). Ambos tipos de esquemas se pueden obtener a partir de la combinación de un esquema de cifrado y un MAC.

118. El modo CCM (*Counter with Cipher Block Chaining-Message Authentication Code*) garantiza la confidencialidad y autenticidad de los datos y se basa en un algoritmo de cifrado en bloque de claves simétricas recomendado cuyo tamaño de bloque sea de 128 bits. Este modo usa conjuntamente los modos CBC-MAC y CTR y ambos se aplican al mismo mensaje; el primero para autenticar el mensaje mediante un MAC y el segundo para cifrarlo. En los dos procesos se usa la misma clave.

119. El modo cifrar-luego-autenticar-luego-traducir o EAX (*Encrypt-then-Authenticate-then-Translate*) es un modo de AEAD que proporciona

autenticación y privacidad del mensaje con un esquema de dos pasadas, una para lograr privacidad y otra para autenticidad para cada bloque.

120. ChaCha20_Poly1305 [RFC8439] es también un modo de AEAD que está compuesto por el cifrador ChaCha20 y el autenticador Poly1305. ChaCha20 es un cifrador de alta velocidad propuesto por Bernstein en [Ber08] y es considerablemente más rápido que AES en implementaciones solo de software. Ello le permite ser tres veces más rápido en plataformas que carecen de hardware AES especializado. Por su parte, Poly1305 es un código de autenticación de mensajes de alta velocidad, también presentado por Bernstein [Ber05].

121. La Tabla 2.9 presenta los cifrados autenticados autorizados con sus principales propiedades.

Esquema	Rec./Her.	Referencias	Notas
Cifrado-y luego-MAC	R	[BN00]	25, 26
CCM	R	[SP800-38C], [ISO19772]	25, 26
GCM	R	[SP800-38D], [ISO19772]	19, 20, 21, 25, 26, 27
EAX	R	[ISO19772]	25, 26
ChaCha20_Poly1305	R	[RFC8439]	26, 28, 29

Tabla 2.9: Esquemas simétricos de cifrado autenticado autorizados

122. Nota 25 **[Esquemas de cifrado autenticado]** Debe tenerse en cuenta que los esquemas de cifrado autenticado simétrico autorizados son todos construcciones criptográficas. Algunos de ellos usan como primitiva un único cifrado en bloque, como CCM o GCM; otros se basan en un esquema de cifrado primitivo y en un esquema de autenticación de mensajes primitivo, que describen solo cómo están compuestos; por ejemplo, Cifrado-y luego-MAC. En tales casos, las primitivas deben estar autorizadas (ver Nota 2). Es importante señalar que las Notas 15 y 16 también se aplican a esquemas de cifrado autenticado simétrico.

123. Nota 26 [**Orden de descifrado.**] Si la integridad de los textos cifrados no se comprueba correctamente antes del descifrado, el desarrollador o evaluador debe asegurarse de que la implementación no cree ninguna instancia de padding u otro oráculo erróneo. Esto significa que la implementación no revela ninguna información sobre el padding o el formato del texto claro obtenido a través del descifrado de un texto cifrado arbitrario. Si no fuera así, un adversario podría descifrar un texto cifrado objetivo explotando, por ejemplo, mensajes de error o información derivada de un ataque por canal lateral por análisis temporal. Los textos claros nunca deben enviarse a las aplicaciones que los vayan a utilizar antes de que se haya verificado su integridad.

124. Nota 27 [**Longitud del texto claro GCM**] Por especificación, en cada invocación de GCM, la longitud del texto claro debe ser, como máximo, de $2^{32} - 2$ bloques del cifrado de bloque subyacente. Es necesario verificar que no se exceda esta longitud máxima en entornos donde esto pudiera suceder.

125. Nota 28 [**Esquema ChaCha20_Poly1305**] El esquema autorizado usa una variante del cifrador ChaCha20 con 20 rondas y una clave de 256 bits. Es sabido que existen variantes con claves de 128 bits y de entre 8 y 12 rondas, pero no esas otras variantes no están autorizadas.

126. Nota 29 [**No reutilización del vector de inicialización**] El vector de inicialización debe procesarse dentro del perímetro de seguridad del esquema de cifrado autenticado. Así, es esencial asegurarse de que un adversario nunca pueda hacer que se use el mismo valor IV para proteger dos pares diferentes de mensaje y datos asociados con la misma clave. La reutilización de un IV puede afectar la confidencialidad.

2.2.6. Protección de las Claves

127. Un mecanismo de protección de claves o de envoltura de claves, permite almacenar o transmitir claves de forma segura, lo que garantiza confidencialidad, integridad y autenticación de los datos de origen de la clave.

128. Un vector de inicialización sintético o SIV (*Synthetic Initialization Vector*) es un modo de operación de cifrado en bloque que considera una clave, un texto claro y múltiples cadenas de octetos de longitud variable que se autenticarán pero no se cifrarán. SIV produce un texto cifrado que tiene la misma longitud que el texto claro y un vector de inicialización sintético. Dependiendo de cómo se utilice, SIV logra el objetivo de cifrado autenticado determinista o el objetivo del cifrado autenticado resistente al uso indebido y basado en *nonce*.

129. AES-Keywrap permite la protección de la confidencialidad y la integridad de las claves criptográficas. En particular, KW (*Key Wrap*) y KWP (*Key Wrap with Padding*) son dos modos de operación deterministas de cifrado autenticado del algoritmo AES.

130. La Tabla 2.10 presenta los esquemas de protección de claves autorizados y sus principales propiedades.

Esquema	Rec./Her.	Referencias	Notas
SIV	R	[RFC5297]	
AES-Keywrap	R	[SP800-38F, Alg. KW & KWP]	

Tabla 2.10: Esquemas de protección de claves autorizados

2.2.7. Funciones de Derivación de Claves

131. Una función de derivación de claves o KDF (*Key Derivation Function*) permite obtener varias claves a partir de una única clave maestra. En general, la función considera como entrada tres argumentos: un valor secreto K, un valor (posiblemente) público N y una longitud n, y genera n bits, que pueden dividirse en varias claves que parecen ser independientes. Hay muchas buenas formas para implementar estas funciones. La lista de mecanismos de derivación de claves autorizados que se proporciona en la Tabla 2.11 no pretende ser exhaustiva. En general, una KDF se considera autorizada si los mecanismos criptográficos que emplea lo están.

132. Los dos primeros grupos de funciones de derivación de claves están estandarizadas por el NIST en los documentos 56A, 56B, 56C y 108 de la familia SP800. Estas KDF se basan en los problemas del logaritmo discreto, de la factorización de enteros y de una extracción seguida de una expansión.

133. La función X9.63-KDF [ANSIX9.63] está estandarizada por ANSI y se fundamenta en funciones resumen.

134. La función 2 de derivación de clave basada en contraseña o PBKDF2 (*Password-Based Key Derivation Function 2*) es una KDF con un costo computacional variable, que se utiliza para reducir las vulnerabilidades de los ataques

de fuerza bruta [SP800-132]. Esta función forma parte de los estándares PKCS de los laboratorios RSA, específicamente PKCS #5 v2.0 [RFC2898], [RFC8018]. No debe confundirse esta PBKDF2 con la PBKDF1, dado que la última solo da lugar a claves derivadas de hasta 160 bits. Para la PBKDF2 se añade una nota específica que los desarrolladores y evaluadores deben tener en cuenta.

135. La función HKDF es una función de derivación de clave simple (KDF) basada en el código de autenticación de mensajes HMAC [RFC5869]. HKDF sigue el protocolo de extraer y luego expandir, donde el KDF consta de dos etapas. En la primera considera la entrada y extrae una clave pseudoaleatoria de longitud fija y en la segunda la expande en varias claves pseudoaleatorias adicionales, que es su salida.

Esquema	Rec./Her.	Referencias	Notas
NIST SP800-56A/B/C	R	[SP800-56A], [SP800-56B], [SP800-56C]	
NIST SP800-108	R	[SP800-108]	
ANSI-X9.63-KDF	R	[ANSIX9.63]	
PBKDF2	R	[RFC8018], [SP800-132]	30
HKDF	R	[RFC5869]	

Tabla 2.11: Funciones de derivación de claves autorizadas[5]

136. Nota 30 **[PBKDF2]** Esta función de derivación de claves se basa en una función pseudoaleatoria, que se puede concretar usando una función de generación de MAC. La función pseudoaleatoria deberá ser un mecanismo autorizado. En el caso de que se utilice una función HMAC, se debe vigilar la longitud de la clave. De hecho, si la clave HMAC es más larga que la longitud del bloque de mensajes de la función resumen, la clave es resumida. Este prerresumen en HMAC puede reducir la entropía efectiva de la clave derivada.

2.2.8. Mecanismos de Protección de Contraseñas

137. Los mecanismos de protección de contraseñas asocian a una contraseña un valor de verificación, de forma que los sistemas de autenticación de usuarios que se basan en contraseñas no almacenan las propias contraseñas, sino estos valores de verificación. De este modo, las contraseñas nunca están disponibles (no se almacenan en claro

[5] La lista de mecanismos de derivación de claves autorizados que se proporciona no pretende ser exhaustiva.

por lo que no estarán comprometidas), pero es posible verificar que pertenecen a determinado usuarios, comprobando su correspondiente valor de verificación.

138. Argon2id (ver [BDK16], [RFC9106] y [AS15]) es una función de *memory-hard* (memoria difícil) diseñada específicamente para ser resistente a ataques de fuerza bruta y de tipo diccionario, siendo especialmente adecuada para la protección de contraseñas. Es parte del framework Argon2, que ganó el concurso Password Hashing Competition en 2015.

139. SCRYPT, por su parte, es una KDF basada en contraseñas propuesta en [Per09] para el servicio de copias de seguridad en línea de Tarsnap. Esta KDF se diseñó para que fuera difícil realizar ataques de hardware personalizados a gran escala, dado que requiere grand cantidad de memoria. De hecho, algunas criptomonedas utilizan una versión simplificada de SCRYPT como esquema de prueba de trabajo.

140. En la Tabla 2.12 se incluyen las características del mecanismo de resumen de contraseñas autorizado.

Esquema	Rec./Her.	Referencias	Notas
Argon2id	R	[RFC9106]	31
PBKDF2	R	[RFC8247]	32, 33
SCRYPT	R	[RFC7914]	

Tabla 2.12: Mecanismos de resumen de contraseñas autorizados

141. Nota 31 **[Uso de Argon2id]** La primera configuración que se recomienda como configuración predeterminada en todos los entornos de Argon2id es la variante con los parámetros $t = 1$ y 2 GB de memoria debido a que es segura contra ataques por canal lateral y maximiza los costos de confrontación en hardware dedicado de fuerza bruta. La segunda opción recomendada es la variante con $t = 3$ y 64 MB de memoria, y se sugiere como configuración predeterminada para entornos con memoria limitada.

142. Nota 32 **[Número de iteraciones]** Los mecanismos de resumen de contraseñas permiten seleccionar parámetros que controlan la complejidad de la ejecución del resumen. De esta manera se incrementa el trabajo para realizar posibles ataques de fuerza bruta. De hecho, un uso legítimo del resumen de una contraseña solo necesita una ejecución, mientras que un ataque de fuerza bruta precisa una gran cantidad de ejecuciones. Así pues, para proteger este mecanismo, el número de iteraciones de PBKDF2 debe seleccionarse lo más grande posible.

Nota 33 [***Salt***] Este proceso consiste en generar un valor aleatorio (*salt*) cuando se registra una contraseña y que se almacena junto con el valor de verificación de dicha contraseña. De este modo, el *Salt* de un mecanismo de resumen de contraseñas permite contrarrestar los ataques por precálculo. La longitud del valor generado debe ser de, al menos, 128 bits.

2.2.9. Mecanismos de Combinación de Claves

143. Un mecanismo de combinación de claves permite juntar claves acordadas usando diferentes métodos de establecimiento de claves (sección 3.2.4). Esto asegura que la creación de la clave resultante sea segura siempre que al menos uno de los métodos utilizados sea robusto.

144. Este mecanismo es necesario para crear los denominados *métodos híbridos de establecimiento de claves*, por ejemplo, combinando un KEM postcuántico con uno clásico basado en EC-DH. Estos mecanismos toman como entrada dos o más claves secretas (y posiblemente además los mensajes intercambiados por los respectivos métodos de establecimiento de claves) y genera una clave combinada como salida.

145. En la Tabla 2.13 se incluyen mecanismos de hibridación de claves autorizados.

Esquema	Rec./Her.	Referencias	Notas
CatKDF	R	[ETS20, §8.2]	
CasKDF	R	[ETS20, §8.3]	

Tabla 2.13: Mecanismos de hibridación de claves autorizados

146. En el anexo B.4.2 se describen en detalle ambos esquemas.

3. Mecanismos Asimétricos

147. Los mecanismos asimétricos son aquellos, como ya se introdujo en la Sección 1, en los que las acciones que llevan a cabo o los parámetros que utiliza cada una de las partes que intervienen son diferentes. En esta sección se abordarán los mecanismos que corresponden a esta categoría.

3.1. Primitivas Asimétricas

148. La criptografía asimétrica (a veces también llamada de clave pública) tiene como propiedad distintiva que cada usuario emplea dos claves, una para el proceso de cifrado y otra diferente para el de descifrado. La primera de las claves es la «clave pública» que cada usuario da a conocer para que sea utilizada como clave para cifrar los mensajes que se le envíen; mientras que la otra es la «clave privada» (o secreta), que solo conoce dicho usuario y le permite descifrar los mensajes cifrados que recibe.

149. Ambas claves están relacionadas mediante un problema matemático y dado que ambas llevan a cabo procesos inversos (una cifra y la otra descifra), tal problema se elige de modo que el primero de los procesos (cifrado) suponga resolver un problema matemático sencillo, a la vez que el segundo proceso (descifrado) equivalga a determinar la solución de un problema matemático computacionalmente imposible de resolver en un tiempo razonable. Es claro que, como las claves están relacionadas, el problema matemático seleccionado debe garantizar que el conocimiento de la clave pública no permite recuperar la clave privada. Así pues, la seguridad de la criptografía asimétrica se basa en la supuesta dificultad de resolver computacionalmente un problema matemático.

150. Debido a que los mecanismos asimétricos hacen uso de la clave privada de un usuario para, principalmente, descifrar un texto cifrado (esquemas de cifrado) o elaborar una firma válida (esquemas de firma electrónica), no debería ser posible realizar ninguna operación que requiera la clave privada con el conocimiento exclusivo de la clave pública. Esta propiedad debe mantenerse incluso en el caso de que un adversario reciba resultados de operaciones privadas que requieren la clave privada, donde los recursos de estas operaciones privadas son conocidos o elegidos por el adversario.

151. Comenzamos presentando en esta sección las primitivas atómicas asimétricas que están aceptadas y los problemas matemáticos correspondientes. En la siguiente sección se mostrarán las construcciones asimétricas construidas a partir de estas primitivas. Debe tenerse en cuenta que solo se consideran aceptadas las primitivas o construcciones cuyos parámetros satisfagan las condiciones que se muestren en las diferentes tablas que se incluyen.

3.1.1. Problema de la Factorización de Números Enteros (RSA)

152. La seguridad de varios esquemas criptográficos asimétricos, incluidos los esquemas de cifrado y firma electrónica basados en el RSA, se basa en la dificultad que supone resolver el problema de la factorización de números enteros (IFP) en comparación con el problema, que puede considerarse su inverso, de la multiplicación de enteros y con el problema de la exponenciación modular. Para los dos últimos existen algoritmos cuyo tiempo es polinómico; mientras que para el primero, el mejor algoritmo conocido requiere de tiempo subexponencial.

153. La primitiva RSA puede considerarse como permutación pública parametrizada por una clave pública y la permutación inversa privada parametrizada por la clave privada asociada. Esta primitiva se utiliza en los esquemas de cifrado y firma de RSA. Tales esquemas especifican, por su parte, aspectos relacionados con la verificación de relleno (*padding*), redundancia, etc. Recordamos que las primitivas por sí solas no debe considerarse como esquemas completos de cifrado o firma, dado que precisan convenciones adicionales para garantizar su seguridad.

154. Sean p y q dos números primos grandes, elegidos al azar, y sea su producto $N = p \cdot q$, que se denotará como el «módulo RSA». El valor $n = \lfloor \log_2(N) \rfloor + 1$ es el tamaño de N, esto es, su longitud en binario. La clave pública es el par formado por el módulo N junto con un elemento e, llamado «exponente público», que es un invertible módulo $\phi = (p-1)(q-1)$, es decir, es primo con ϕ, o lo que es igual, $\mathrm{mcd}\,(e, \phi) = 1$. El inverso de e módulo $\mathrm{mcm}\,(p-1, q-1)$, denotado por d, se llama «exponente privado». La clave privada está formada por este exponente privado junto con el módulo.

155. La permutación pública mencionada en el párrafo anterior opera sobre los números enteros módulo N, $\mathbb{Z}_N$, y consiste en la exponenciación de la entrada considerada elevada a la potencia e, módulo N. Recuérdese que tanto N como e son públicos, por lo que cualquiera que los conozca puede determinar, fácilmente, el valor resultante de la expresión

$$M^e \pmod{N}, \qquad \text{donde } M \in \mathbb{Z}_N.$$

Por otra parte, la permutación privada opera sobre el conjunto de los números enteros módulo N, $\mathbb{Z}_N$, y consiste en la exponenciación de la entrada elevada a la potencia d, módulo N, esto es,

$$C^d \pmod{N}, \qquad \text{siendo } C \in \mathbb{Z}_N.$$

En este caso, como el valor de d solo es conocido por el propietario de la clave privada, nadie, salvo él, puede determinar el valor de la expresión anterior.

156. En la Tabla 3.1 se muestran los tamaños de las claves autorizadas para la primitiva basada en el problema de la factorización de números enteros (RSA) y otras características.

Primitiva	Tamaño de los parámetros (bits)	Rec./Her.	Referencias	Notas
RSA	$n \geq 3000, \log_2(e) > 16$ $n \geq 1900, \log_2(e) > 16$	R H[2025]	[RSA78]	34

Tabla 3.1: Tamaño de las Primitivas RSA autorizadas

157. Nota 34 **[RSA Heredado]** La fecha límite aceptable para el uso heredado de un módulo de tamaño superior a 1900 bits, pero inferior a 3000 bits, se establece en el 31 de diciembre de 2025.

3.1.2. Problema del Logaritmo Discreto Multiplicativo

158. De forma análoga a como sucede con el problema de la factorización de números enteros y de su problema inverso, la multiplicación de enteros (ver §3.1.1), la seguridad de otros esquemas criptográficos asimétricos se basa en la dificultad de resolver el problema del logaritmo discreto (DLP, *Discrete Logarithm Problem*) en el grupo multiplicativo de un cuerpo finito, en contraposición con la facilidad del problema de la exponenciación modular (MODP, *modular exponentiation*), también en un cuerpo finito.

159. Es posible elegir diferentes cuerpos finitos en los que implementar este problema, pero la solución más segura y ampliamente utilizada es considerar un cuerpo finito primo, $\mathbb{F}_p \approx GF(p)$, siendo p un número primo grande. En esta guía, y mientras no se diga lo contrario, se supondrá que el cuerpo considerado es de este tipo.

160. Esta primitiva cuya seguridad se basa en el problema del logaritmo discreto en el grupo multiplicativo de $\mathbb{F}_p$ se utiliza en varios esquemas de acuerdo o intercambio de claves y firmas. Sea g un generador de un subgrupo de orden q del grupo multiplicativo $\mathbb{F}_p^*$, donde q divide a $p-1$ dado que $p-1 = |\mathbb{F}_p^*|$, y sea r el factor primo más grande de q. La primitiva es la función de exponenciación de base g en $\mathbb{F}_p$ de modo que si se considera como entrada el entero x, en general con $1 \leq x \leq q-1$, proporciona como salida el entero $y = g^x$. Según como el esquema considerado utilice esta primitiva, x e y pueden representar (una parte de) una clave privada y la clave pública asociada, o pueden representar un exponente efímero de Diffie-Hellman y su valor público asociado, etc. En general, el mecanismo de intercambio de claves efímeras de Diffie-Hellman basado en cuerpos finitos se representa por FFDHE (*Finite-Field-based Diffie-Hellman Ephemeral key exchange mechanism*).

161. La Tabla 3.2 presenta los tamaños de las claves autorizadas para la primitiva basada en el problema del logaritmo discreto multiplicativo sobre un cuerpo finito primo,

cuya primitiva asociada se abrevia como FF-DLOG (*Finite Field-Discrete Logarithm*) y sus características. Por otra parte, para todos los grupos considerados, se tiene que $r = q = (p-1)/2$.

Familia	Tamaño del grupo (bits)	Rec./Her.	Referencias	Notas
MODP	3072 bits	R	[RFC3526]	
	4096 bits	R		
	6144 bits	R		
	8192 bits	R		
	2048 bits	H[2025]		35, 36
FFDHE	3072 bits	R	[RFC7919]	
	4096 bits	R		
	6144 bits	R		
	8192 bits	R		
	2048 bits	H[2025]		35, 36

Tabla 3.2: Tamaño de las primitivas autorizadas del logaritmo discreto multiplicativo sobre un cuerpo finito

162. En el caso de que no se empleen los parámetros señalados en la Tabla 3.2, existe la posibilidad de generar nuevos grupos, siempre que se tengan en cuenta los esquemas incluidos en la Tabla 3.3.

Esquema	Rec./Her.	Referencias
Generación de parámetros (p, q)	R	[FIPS186-4, App. A.1.1.2]
Generación de g	R	[FIPS186-4, App. A.2.3]

Tabla 3.3: Esquemas autorizados para generar nuevos grupos

163. Estos métodos generan un subgrupo de orden primo, es decir, q es primo y por tanto $r = q$. Se autorizan los siguientes tamaños de parámetros mostrados en la Tabla 3.4.

Primitiva	Tamaño de parámetros	Rec./Her.	Notas
FF-DLOG	$\log_2(p) \geq 3000, \log_2(q) \geq 250$	R	35, 36
	$\log_2(p) \geq 1900, \log_2(q) \geq 200$	H[2025]	

Tabla 3.4: Parámetros autorizados para generar nuevos grupos

164. Nota 35 [**Precomputación**] Los algoritmos de logaritmos discretos implican una fase de precomputación relacionada con el grupo, que es el cuello de botella en términos de complejidad del ataque. Como consecuencia, para los módulos del logaritmo discreto compartidos por muchos usuarios y aplicaciones, se recomienda encarecidamente no utilizar módulos de longitud cercana al límite inferior del rango heredado.

165. Nota 36 [**DLP Heredado**] La fecha límite de aceptación para el uso heredado de grupos multiplicativos módulo un primo de tamaño superior a 1900 bits, pero inferior a 3000 bits, se establece en el 31 de diciembre de 2025.

3.1.3. Problema del Logaritmo Discreto Aditivo

166. La dificultad del problema del logaritmo discreto también se puede definir en el grupo de puntos racionales de una curva elíptica definida sobre un cuerpo finito. En este caso, el problema se denomina problema del logaritmo discreto sobre curvas elípticas, elíptico, aditivo o ECDLP (*Elliptic Curve Discrete Logarithm Problem*), en contraposición al caso anterior, en el que la operación considerada era la multiplicación en un grupo finito. En este caso, la primitiva se conoce como logaritmo discreto sobre curvas elípticas y de forma abreviada como EC-DLOG (*Elliptic Curve Discrete Logarithm*).

167. Para comprender bien la primitiva asociada a este problema, conviene considerar la siguiente notación y recordar las siguientes propiedades.

168. Sea p un número primo y $\mathbb{F}_p$ el cuerpo primo con p elementos. Sea, además, $E(\mathbb{F}_p)$ una curva elíptica definida sobre $\mathbb{F}_p$, denotada por E. Dicha curva está definida por una ecuación general del tipo[6]

$$E\colon y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6,$$

donde $a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}_p$.

169. La curva E está formada por los puntos del plano $\mathbb{F}_p \times \mathbb{F}_p$ que verifican dicha ecuación y sobre este conjunto se define una operación de suma de puntos que, junto con el punto del infinito, $\mathcal{O}$, hace que E sea un grupo abeliano (para un estudio más exhaustivo de la criptografía basada en curvas elípticas, recomendamos al lector [GHM18]).

[6] Según sea el cuerpo considerado y su característica, la curva elíptica puede tomar expresiones más sencillas a la dada aquí, de modo que los coeficientes verificarán determinadas condiciones.

170. Sea P un punto de orden q de la curva E, esto es, tal que $qP = P + \overset{(q-2)}{\cdots} + P = \mathcal{O}$. Sea además, r el factor primo más grande de q. La primitiva asociada al problema del logaritmo discreto en E es la multiplicación de puntos de la curva por escalares, de modo que si se considera como entrada el entero x, con $1 \leq x \leq q-1$, la salida es el punto de la curva E dado por $Q = xP$.

171. El orden de la curva, esto es, el número de puntos de la curva, $\#E$, puede ser un número primo o un número compuesto. Al cociente del orden de la curva entre su mayor factor primo se le denomina cofactor, de manera que si G es un punto de orden n que genera un subgrupo cíclico de orden primo, y no existe otro factor primo de $\#E$ mayor que n, entonces se cumplirá que el cofactor, h, de la curva será $h = \#E/n$. En general se recomienda utilizar curvas cuyo orden sea un número primo o que, como segunda opción, sea el producto de un número primo y un cofactor pequeño (típicamente 2, 3 o 4) [HMV04].

172. El conjunto de parámetros públicos de los esquemas criptográficos donde se usa esta primitiva es $\{p, E, P, q\}$. En función del esquema criptográfico que se considere, x y Q pueden representar (una parte de) una clave privada y la clave pública asociada, o pueden representar un valor secreto efímero de Diffie-Hellman y su valor público asociado, etc.

173. En estos grupos, el problema del logaritmo discreto también se considera difícil, en comparación con su operación inversa que, como hemos visto, es la multiplicación de puntos de la curva por escalares. Por otra parte, en este caso es posible seleccionar dos parámetros: el cuerpo finito sobre el cual se definirá la curva elíptica y la propia curva elíptica. También, como en el caso del logaritmo discreto multiplicativo, solo se consideran curvas elípticas definidas sobre cuerpos primos.

174. En la literatura se han propuesto numerosas curvas elípticas, buscando, sobre todo, la sencillez de sus ecuaciones, y cuerpos finitos en los que su cardinal, es decir, el primo considerado, tenga una expresión binaria en la que abunden los ceros, de modo que la implementación de la aritmética de la curva sea eficiente.

175. En la Tabla 3.5 se muestran las familias de curvas elípticas autorizadas para su implementación, así como el nombre correspondiente de cada una de ellas.

Familia de curvas	Curva	Rec./Her.	Referencias	Notas
Brainpool	BrainpoolP256r1 BrainpoolP384r1 BrainpoolP512r1	R R R	[RFC5639]	37, 38, 39
NIST	NIST P-256 NIST P-384 NIST P-521	R R R	[SP800-186]	37, 38, 39, 40
FR	FRP256v1	R	[ANS11]	37, 38, 39
Montgomery	Curve25519 Curve448	R R	[SP800-186]	37, 38, 39
Twisted Edwards	Edwards25519 Edwards448	R R	[SP800-186]	37, 38, 39

Tabla 3.5: Curvas elípticas autorizadas

176. Nota 37 **[Puntos en la curva]** Es preciso verificar que los puntos considerados están en la curva, es decir, verifican su ecuación.

177. Nota 38 **[Puntos en un subgrupo]** Es preciso verificar que los puntos considerados están en el subgrupo considerado de la curva. Nótese que el uso de subgrupos de la curva tiene lugar cuando el orden de la misma no es un número primo.

178. Nota 39 **[Orden primo]** Si el orden del subgrupo es un número primo, esto es, $q = r$, y es tal que r^2 no divide al cardinal de la curva, $\#E(\mathbb{F}_p)$, las comprobaciones mencionadas en la Nota 38 se reducen a verificar que los puntos considerados tienen orden precisamente r.

179. Nota 40 **[Elección del primo]** La especial forma del número primo p seleccionado para la construcción del cuerpo finito $\mathbb{F}_p$ hace que los ataques por canal lateral sean más eficientes que con un primo aleatorio.

180. Nota 41 **[Respecto a las funciones de codificación]** Con relación a los parámetros de las curvas elípticas Curve25519 y Curve448, cuando se realiza una multiplicación escalar de un punto Q por un entero secreto x, se debe comprobar que Q es un múltiplo del punto base P, donde x se genera mediante la aplicación de la función de decodificación definida en [RFC7748, §5] y, además, que el resultado de la multiplicación escalar no es un valor nulo.

3.1.4. Otros Problemas Computacionalemente Difíciles

181. Los tres problemas matemáticos considerados anteriormente (factorización de números enteros, logaritmo discreto multiplicativo y logaritmo discreto aditivo) son los más estudiados y los más utilizados en la criptografía actual. Sin embargo, tal y como se comentará en el Anexo B, estos algoritmos podrían ser considerados vulnerables a la computación cuántica si se utilizara el algoritmo de Shor [Sho97] y se dispusiera de un ordenador cuántico con la suficiente capacidad de cómputo.

182. En este sentido es importante mencionar algunos problemas matemáticos que permiten definir construcciones asimétricas para los cuales no se conoce un método genérico de resolución eficiente en general, ni siquiera mediante el uso de ordenadores cuánticos.

183. En el Anexo B se detallarán los tipos de criptografía que están teniendo más relevancia en la actualidad:

- Basadas en funciones resumen (*hash function*).
- Basada en retículos.

184. No obstante, conviene mencionar que estos ≪nuevos problemas≫ han sido mucho menos estudiados que los tres mencionados anteriormente por lo que solo las primitivas basadas en estos últimos son las autorizadas. Los restantes problemas mencionados están siendo escrutados por la convocatoria internacional hecha por el NIST [NIS17] en busca de estándares que se pretende sean resistentes a la computación cuántica (*quantum resistant*).

3.2. Construcciones Asimétricas

185. Determinados problemas matemáticos, computacionalmente difíciles[7], suelen utilizarse para construir esquemas asimétricos.

186. En los esquemas asimétricos, cada usuario posee un par de claves. La primera de ellas es una clave públicamente conocida, denotada por p_k, y una segunda clave privada, esto es que se mantiene en secreto, designada como s_k. La seguridad de tal esquema debe basarse en la dificultad computacional de un problema matemático, esto es, que conocida la clave pública, determinar la clave privada asociada sea equivalente a resolver dicho problema matemático. Ello supone que no es posible descifrar los mensajes destinados a un usuario que se hayan cifrado con su clave pública, salvo, claro está, el propietario de la clave privada asociada a tal clave pública. De forma análoga, nadie podrá firmar digitalmente un fichero haciendo uso de la clave privada, aunque sí podrá verificar que dicha firma corresponde a tal usuario para lo que utilizarán su correspondiente clave pública.

187. Al margen de que la seguridad de los esquemas asimétricos se base en la seguridad computacional de un problema matemático, también es posible que tal seguridad dependa de la seguridad de un esquema o primitiva simétrico. En todo caso, para que una construcción criptográfica esté autorizada, debe basarse en primitivas también autorizadas. De forma más concreta, los requisitos sobre los tamaños de los parámetros establecidos en la sección anterior (ver §3.1) también se aplican a los esquemas de esta sección.

188. Al margen de lo ya dicho en los párrafos anteriores, es claro que la seguridad de los esquemas asimétricos con clave también requiere de la confidencialidad e integridad de la clave privada y de la integridad y autenticidad del origen de datos de la clave pública.

189. Para cada esquema asimétrico con clave considerado en esta sección, los aspectos específicos de la generación de pares de claves se tratan en la misma subsección que el resto del esquema.

190. En el capítulo 6 se tratará el tema de la gestión de las claves y de los mecanismos para la generación de pares de claves asimétricas autorizadas.

3.2.1. Esquemas de Cifrado Asimétrico

191. Un esquema de cifrado (o criptosistema) asimétrico consta de tres protocolos. El de generación del par de claves; el protocolo de cifrado por el que un mensaje o

[7] Se dice que un problema matemático es computacionalmente difícil si o bien no se conoce un algoritmo que lo resuelva, o si existiendo tal algoritmo, el tiempo de computación necesario para obtener la solución requiere de un tiempo (sub)exponencial, utilizando la mejor tecnología disponible y el algoritmo más eficiente.

texto en claro dado, m, se transforma en un texto cifrado o criptograma mediante la clave pública, p_k; y el protocolo de descifrado, que permite recuperar el texto claro, m, a partir del texto cifrado, c, mediante la clave privada, s_k.

192. El *padding* de cifrado asimétrico óptimo u OAEP (*Optimal Asymmetric Encryption Padding*) es un esquema de relleno que permite completar la longitud necesaria (generalmente en bits) de la entrada a un determinado algoritmo. En general, se utiliza junto con el cifrado RSA. Este OAEP fue introducido en [BR95] y más tarde estandarizado en [RSA12] y en [RFC8017].

193. Los estándares de criptografía de clave pública PKCS son una colección de estándares (desde el PKCS#1 al PKCS#15) desarrollados y publicados por los laboratorios RSA (https://www.rsa.com/en-us/index, https://web.archive.org/web/20061209135809/http://www.rsasecurity.com/rsalabs/node.asp?id=2124).

194. En la Tabla 3.6 se incluyen las características del esquema de cifrado asimétrico RSA autorizado.

Primitiva	Esquema	Rec./Her.	Referencias	Notas
RSA	OAEP PKCS#1 v2.1	$R^{\varnothing}$	[RFC8017], [RSA12]	42, 43, 45
RSA	PKCS#1 v1.5	H	[RFC8017], [RSA12]	42, 44, 45

Tabla 3.6: Esquema de cifrado asimétrico autorizado

195. Nota 42 [**Relleno (*Padding*) aleatorio**] Los esquemas de cifrado asimétrico utilizan un *padding* aleatorio que será generado por un generador de bits aleatorios autorizado (ver el capítulo 5).

196. Nota 43 [**Ataque de relleno-OAEP**] En el caso de que el procedimiento de descifrado OAEP no se implemente de modo correcto, es decir, si las comprobaciones realizadas por la decodificación EME-OAEP no se realizan en el orden especificado, el RSA OAEP puede ser vulnerable a ataques de oráculo.

197. Nota 44 [**Ataque de *padding***] Si hubiera un oráculo de *padding* disponible, el esquema RSA-PCKS #1v1.5 sería vulnerable a ataques eficientes.

198. Nota 45 [**Amenaza de la Computación Cuántica**] RSA es vulnerable a ataques de complejidad polinómica en el modelo de computación cuántica, por ejemplo, el algoritmo de Shor. Tales ataques pueden realizarse de manera retroactiva": el cifrado puede almacenarse ahora y descifrarse más tarde cuando un ordenador cuántico esté disponible. En contextos donde se requiere resistencia contra ataques que utilizan ordenadores cuánticos, no se debe usar este mecanismo sin combinarlo con un mecanismo resistente a la computación cuántica. Los algoritmos que requieren hibridación se muestran con el símbolo $R^{\varnothing}$

3.2.2. Firmas Digitales

199. Los esquemas de firma digital permiten realizar la firma de un documento y constan de tres protocolos: uno de generación del par de claves (pública y privada); otro de elaboración de la firma, cuyas entradas son la clave privada del firmante y el mensaje a firmar) y cuya salida es la firma del mensaje; y un protocolo de verificación de la firma, cuyas entradas son la clave pública del firmante, el mensaje firmado y la firma, siendo su salida o Verdadero o Falso. Los esquemas de firma digital garantizan la autenticación de los datos y el no repudio.

200. En la presente versión de esta guía se incluyen nuevos algoritmos de firma digital resistentes a la computación cuántica. Sin embargo, los ataques a la autenticación en los protocolos basados en firma digital no se pueden realizar de manera retroactiva. Actualmente, el CCN no considera urgente implementar medidas de protección contra la computación cuántica excepto en el caso de firmas digitales de firmware (FW) o software (SW) para aquellos productos en los que no sea fácil actualizar esta verificación.

201. Además, se considera que actualmente las especificaciones de X509 para la gestión de certificados basados en algoritmos de firma digital postcuánticos y que tengan en cuenta la hibridación de distintos sistemas de firma no se encuentran en un estado de madurez suficiente. Por ello cuando exista el requisito de resistencia a la computación cuántica, en este momento solo se considera necesario proteger adecuadamente la firma de FW/SW, con XMSS o con ML-DSA hibridado con un esquema clásico.

202. En la Tabla 3.7 se incluyen las características de los esquemas de firma digital autorizados, donde PSS hace referencia a Esquema de firma probabilística (Probabilistic Signature Scheme).

Primitiva	Esquema	Rec./Her.	Referencias	Notas
RSA	PSS PKCS#1v2.2	R	[RFC8017],[ISO9796-2],[RSA12]	46,47
FF-DLOG	KCDSA Schnorr DSA	R	[ISO14888-3] [ISO14888-3] [FIPS186-4], [ISO14888-3]	46,47,48 46,47,48 46,47,48
EC-DLOG	EC-KCDSA EC-DSA EC-GDSA EC-Schnorr EdDSA	R	[ISO14888-3] [FIPS186-5], [ISO14888-3] [TR-03111v2.1] [ISO14888-3] [FIPS186-5]	46,47,48 46,47,48 46,47,48 46,47,48 46,47
RSA	PKCS#1v1.5	H	[RFC8017],[ISO9796-2],[RSA12]	46,47,49
Hash	XMSS	R	[RFC8391], [SP800-208]	50,51, 52,53,54
Retículo	ML-DSA	R	[FIPS204]	55,56
Hash	SLH-DSA	R	[FIPS205]	55

Tabla 3.7: Esquemas de firma digital autorizados

203. Nota 46 [**Función resumen**] El esquema estará autorizado siempre que la función resumen subyacente lo esté (ver §2.1.3).

204. Nota 47 [**Problema difícil**] El esquema estará autorizado siempre que el problema matemático subyacente utilice los parámetros autorizados (ver §3.1).

205. Nota 48 [**Aleatoriedad DSA**] En el algoritmo DSA y en sus variantes de curva elíptica, el procedimiento de elaboración de firmas genera un valor aleatorio. La exfiltración de los valores aleatorios por firma utilizados durante la elaboración de las firmas plantea riesgos, a largo plazo, para la confidencialidad de las claves asociadas. Por lo tanto, debe evitarse tal exfiltración. En principio, esto se refiere tanto a la filtración estadística a través de sesgos en el generador de números aleatorios utilizado como a las filtraciones en el valor de bits aleatorios particulares que puede obtener un atacante (por ejemplo, a través del análisis de canales laterales). Por lo tanto, se recomienda utilizar un generador de números aleatorios con un fuerte postprocesamiento criptográfico, seguridad hacia atrás mejorada y reelección de la semilla desde una fuente verdaderamente aleatoria (ver §5.1).

206. Nota 49 [**Verificación de formato PKCS**] Las comprobaciones de formato deben implementarse cuidadosamente para evitar los ataques de Bleichenbacher [Ble98].

207. Nota 50 [**Firmas digitales postcuánticas con estado**] Se acepta el uso de XMSS únicamente para verificación de FW/SW.

208. Nota 51 [**Firmas digitales postcuánticas con estado**] En XMSS la clave privada se actualiza cada vez que se firma un mensaje. Si la clave se almacena en memoria no volátil debe asegurarse que la clave privada se actualiza en la memoria no volátil antes de exportar la firma correspondiente.

209. Nota 52 [**Precauciones copias de seguridad XMSS**] La seguridad de las firmas XMSS se debilita cuando se utiliza la misma clave privada y el mismo contador para firmar documentos distintos. Se deben tomar las precauciones adecuadas para evitar que suceda este problema durante el proceso de recuperación de una copia de seguridad.

210. Nota 53 [**Firmas digitales postcuánticas con estado**] Se debe comprobar, antes de generar la firma de un mensaje, que el contador de la clave privada no sobrepasa el máximo permitido por la parametrización establecida.

211. Nota 54 [**Firmas digitales postcuánticas con estado**] Se debe verificar que la derivación de las claves WOTS$^+$ se realice siguiendo lo indicado en la [SP800-208], es decir, utilizando la función "PRF_keygen" para evitar "*multi-target attacks*".

212. Nota 55 [**Firmas digitales postcuánticas**] Los mecanismos de firma digital estandarizados por el NIST en 2024 [FIPS204] [FIPS205] se consideran seguros, esto es, se puede afirmar que, hasta la fecha, no se conocen ataques que puedan vulnerar su seguridad, ni siquiera recurriendo a la potencia de cómputo de ordenadores cuánticos.

213. Nota 56 [**Hibridación de firmas basadas en retículos**] Los mecanismos criptográficos basados en retículos deben utilizarse en combinación con mecanismos criptográficos clásicos. La hibridación de firmas digitales puede consistir en concatenar las firmas de diferentes esquemas, de manera que la función de verificación se acepte si y solo si todas las firmas de los distintos esquemas son correctas.

214. Merecen mención especial los algoritmos de firma con resistencia a la computación cuántica que se han añadido recientemente: XMSS, ML-DSA y SLH-DSA.

215. **Algoritmo XMSS**

216. Este algoritmo (véase el anexo B.3.2) fue propuesto por Buchmann et al. [BDH11] y desarrollado posteriormente en [RFC8391] y [SP800-208], como un algoritmo extendido del esquema de firma de Merkle o MSS (*Merkle Signature Scheme*) [Mer89].

217. Está recomendado únicamente para ciertos casos de uso como es la firma de Firmware y Software por parte de las raíces de confianza (Root of Trust). Las peculiaridades de su implementación no lo hacen recomendable para un uso general como algoritmo de firma, ya que cada clave puede emitir un número limitado de

firmas y requiere gestionar su estado interno para evitar duplicidades de uso. Para más detalles sobre el funcionamiento de este algoritmo véase §B.3.2.

218. **Algoritmo ML-DSA**

219. En agosto de 2024 el NIST publicó la versión definitiva del estándar del **algoritmo ML-DSA** [FIPS204], *Module-Lattice-Based Key-Encapsulation Mechanism Standard*. El algoritmo ML-DSA se basa en el algoritmo CRYSTALS-Dilithium [LDK+20], que fue seleccionado por el NIST en 2022 [NIS22] como mecanismo de firma digital y basa su seguridad en problemas definidos sobre retículos. Una descripción detallada del algoritmo se presenta en el anexo dedicado a la criptografía postcuántica (véase B.2.4).

220. **Algoritmo SLH-DSA**

221. También en agosto de 2024, el NIST publicó el estándar del **algoritmo SLH-DSA** [FIPS205], *Stateless Hash-Based Digital Signature Standard*. El algoritmo SLH-DSA se definió apartir del algoritmo SPHINCS+ [HBD+20], cuya seguridad se basa en una función hash sin estado. Puede analizarse con más detalle en el anexo dedicado a la criptografía postcuántica (véase B.3.1).

222. Respecto a las **firmas digitales no resistentes a la computación cuántica** podemos encontrar una descripción detallada de los algoritmos en el anexo C.

3.2.3. Esquemas de Autenticación de Entidad Asimétrica

223. Los esquemas de autenticación de entidades asimétricas permiten que una entidad pruebe su identidad ante otra, demostrando su conocimiento de una clave privada. Estos esquemas son esquemas interactivos por naturaleza y generalmente consisten en utilizar un esquema de firma en un protocolo de desafío-respuesta aleatorio. En esta subsección no se proporciona ningún listado autorizado de estos esquema porque, aunque pueden basarse en esquemas de firma, son un tipo distinto de esquemas, con diferentes objetivos de seguridad. Por lo tanto, la misma clave no debe ser utilizada por un esquema de firma y por un esquema de autenticación de entidad asimétrica (ver Nota 102).

224. En cuanto a los esquemas de autenticación de entidades simétricas, el desafío debe verificar algunas propiedades como las señaladas en la Nota 24.

3.2.4. Establecimiento de Claves y Encapsulación de Claves

225. Los esquemas de establecimiento de claves asimétricas permiten que dos o más partes generen un secreto común sin utilizar ningún valor secreto previamente compartido. Por lo general, estos esquemas se combinan con los de autenticación

asimétrica o simétrica basada en un par de claves público-privada o una clave secreta compartida. El esquema de establecimiento de claves entre dos partes más utilizado es el de Diffie y Hellman [DH76] y se basa en el problema del logaritmo discreto, ya sea utilizando la aritmética multiplicativa (grupo multiplicativo de un cuerpo finito $\mathbb{F}_p^*$), en cuyo caso se denota simplemente como DH, o la aditiva (grupo de los puntos de una curva elíptica definida sobre un cuerpo finito, $E(\mathbb{F}_q)$), denotándose entonces como EC-DH (*Elliptic Curve-Diffie-Hellman*). En cualquiera de los casos, se procede de la siguiente manera:

1. Ambos usuarios, $\mathcal{A}$ y $\mathcal{B}$, acuerdan un grupo cíclico finito, G, y un generador del mismo g.

2. Cada usuario genera un valor aleatorio v_i, $i \in \{A, B\}$ y envía al otro usuario el valor g^{v_i} en el caso multiplicativo y $v_i \cdot g$ en el caso aditivo.

3. Ambos usuarios pueden entonces calcular el elemento común del grupo, que vendrá dado como $(g^{v_A})^{v_B} = (g^{v_B})^{v_A} = g^{v_A v_B}$ en el caso multiplicativo y como $v_A(v_B \cdot g) = v_B(v_A \cdot g) = (v_A v_B)g$ en el caso aditivo. Es claro que cada uno de ellos puede calcular dicho valor a partir del propio valor aleatorio y del elemento recibido del otro usuario.

226. Por otro lado, los métodos de encapsulación de claves (KEM) ofrecen una alternativa para la compartición de claves entre dos partes. El destinatario comienza generando un par de claves (sk, pk). El remitente luego, para la clave pública pk, puede encapsular una clave secreta K en un texto cifrado C. Finalmente, el destinatario puede desencapsular la clave secreta K del texto cifrado C, utilizando su clave privada sk.

227. De forma más precisa, un KEM consta de los siguientes tres algoritmos:

1. KEM.$\mathcal{G}()$: un algoritmo de generación de claves que proporciona como salida un par de claves pública-privada, (pk, sk), cuya estructura depende del esquema en particular que se considere.

2. KEM.$\mathcal{E}c(pk)$: un algoritmo de encapsulado que considera como entrada una clave pública, pk, y genera como salida el par formado por una clave secreta y un texto cifrado (K, C).

3. KEM.$\mathcal{D}c(sk, C)$: un algoritmo de desencapsulado que toma como entrada una clave privada, sk, y un texto cifrado, C, proporcionando como salida una clave secreta, K.

228. Es importante destacar que estos protocolos son vulnerables a los ataques de MitM. En particular, se deben realizar pasos adicionales y se deben intercambiar datos adicionales para garantizar la autenticación de los usuarios y de los mensajes de establecimiento de claves.

229. En la Tabla 3.8 se muestran los esquemas de establecimiento de claves autorizados y sus características más relevantes.

Primitiva	Esquema	Rec./Her.	Referencias	Notas
FF-DLOG	DH DLIES-KEM	$R^{\varnothing}$	[ISO11770-3], [SP800-56A] [ISO18033-2]	57,58,63
EC-DLOG	EC-DH ECIES-KEM	$R^{\varnothing}$	[ISO11770-3], [SP800-56A] [ISO18033-2]	57,58,63
Retículo	ML-KEM FrodoKEM	R	[FIPS203] [FRODO-KEM]	57,59,60 57,59,61,62

Tabla 3.8: Esquemas de establecimiento de claves y de encapsulación de claves autorizados

230. Nota 57 [**Autenticación**] Estos protocolos no están autenticados y pueden ser víctima de ataques MitM. Con el fin de garantizar su seguridad, es preciso autenticar a las dos partes y los datos intercambiados durante el proceso. Esta autenticación requiere secretos a largo plazo.

231. Nota 58 [**Ataques al subgrupo DH/EC-DH**] Tanto si el protocolo Diffie-Hellman se define sobre el grupo multiplicativo de un cuerpo finito (DH) o sobre el grupo de puntos racionales de una curva elíptica definida sobre un cuerpo finito (EC-DH), debe asegurarse que los valores manipulados se encuentren en el subgrupo deseado y tengan un orden lo suficientemente grande (ver Notas 37 y 38).

232. Nota 59 [**Hibridación de claves**] Los mecanismos criptográficos basados en retículos deben utilizarse en combinación con mecanismos criptográficos clásicos. Ver sección 2.2.9.

233. Nota 60 [**ML-KEM**] Es preferible utilizar la versión ML-KEM-1024. Si no es posible, la versión ML-KEM-768 también está aceptada.

234. Nota 61 [**Parámetros de FrodoKEM**] Es preferible utilizar la versión (e)FrodoKEM-1344. Si no es posible, la versión (e)FrodoKEM-976 también está aceptada. Se prefiere el modo efímero dentro de una versión específica.

235. Nota 62 [**Implementación FrodoKEM**] Se deberá seguir la implementación descrita en [ISO18033-2] una vez que se publique definitivamente (actualmente está en desarrollo la versión Amendment 2).

236. Nota 63 [**Amenaza de la Computación Cuántica**] Existen algoritmos que aprovechan los ordenadores cuánticos, como por ejemplo el algoritmo de Shor, que rompen completamente las primitivas asimétricas en las que se basan estas construcciones criptográficas. Dichos ataques pueden realizarse de manera "retroactiva": el cifrado puede almacenarse ahora y descifrarse más tarde cuando un ordenador cuántico esté disponible. En contextos donde se requiere resistencia contra ataques que utilizan ordenadores cuánticos, estos mecanismos criptográficos no deben usarse sin combinarse con un mecanismo resistente a la computación cuántica. Los algoritmos que requieren hibridación se muestran con el símbolo $R^{\varnothing}$.

237. **Algoritmo ML-KEM**

238. En agosto de 2024 el NIST publicó la versión definitiva del estándar del algoritmo ML-KEM [FIPS203], *Module-Lattice-Based Key-Encapsulation Mechanism Standard*. El algoritmo ML-KEM se basa en el algoritmo CRYSTALS-Kyber [SAB+20], que fue seleccionado por el NIST en 2022 [NIS22] como mecanismo de encapsulamiento de claves resistente a la computación cuántica. Su seguridad se basa en problemas definidos sobre retículos. Para una información más detallada sobre sus propiedades y características, remitimos al lector al capítulo donde se trata la criptografía postcuántica, en particular, véase §B.2.2.

239. **Algoritmo FrodoKEM**

240. El algoritmo basado en retículos no estructurados denominado FrodoKEM [FRODO-KEM] se puede considerar como una opción conservadora con relación a su seguridad. Debe tenerse en cuenta que FrodoKEM fue incluido por el NIST en la tercera ronda como un algoritmo alternativo y no como finalista, habiendo sido descartado a partir de la tercera ronda [NIS20].

241. Las razones que el NIST alegó para su decisión se deben, en gran parte, a que su rendimiento es menor que el de otros algoritmos basados en retículos. Este menor rendimiento se debe a que FrodoKEM no emplea ninguna estructura matemática adicional, al contrario de lo que sucede con otros algoritmos basados en retículos. Esta falta de estructura subyacente hace que FrodoKEM sea la opción de seguridad más conservadora, de ahí que el CCN, al igual que otros organismos de seguridad europeos, lo mantenga como algoritmo autorizado para KEM.

4. Protocolos Criptográficos

242. El protocolo TLS o protocolo de seguridad de la capa de transporte, tiene como objetivo la protección de las comunicaciones llevadas a cabo a través de Internet mediante la configuración de los canales que emplean cliente y servidor.

4.1. TLS

243. El protocolo de seguridad en la capa de transporte o protocolo TLS (*Transport Layer Security*), anteriormente denominado protocolo de la Capa de conexión segura o protocolo SSL[8] (*Secure Socket Layer*), es un protocolo que permite proteger las comunicaciones que se realizan a través de Internet [RFC5246], como puede ser, por ejemplo, la conexión a través del protocolo seguro de transferencia de hipertexto o HTTPS (*Hypertext Transfer Protocol Secure*) o el protocolo seguro de transferencia de archivos o FTPS (*File Transfer Protocol Secure*). En definitiva, el protocolo TLS permite configurar y utilizar canales seguros entre las dos partes de una conexión: cliente y servidor [ANS20b], [TR-02102-2].

244. Nota 64 [**Versiones de SSL**] Las versiones v2 y v3 de SSL no están recomendadas.

245. Nota 65 [**Versiones de TLS**] Las versiones 1.0 y 1.1 de TLS no están recomendadas.

246. No obstante, antes de que se puedan transmitir los datos, se debe establecer un canal o conexión segura entre el cliente y el servidor. Este proceso se denomina protocolo de enlace o *handshake* y es una parte importante del protocolo TLS. En este protocolo, el cliente y el servidor acuerdan

- Los algoritmos criptográficos que emplearán para el cifrado de datos, la protección de la integridad, el acuerdo de claves y, si es necesario, la autenticación entre las partes, ya sea unilateral (en general, el servidor se autentica ante el cliente) o bilateral (ambas partes se autentican entre sí).
- Un secreto compartido, es decir, un secreto maestro, del que se derivarán las claves de sesión que usarán posteriormente para la protección de la integridad y para el cifrado de datos.

[8] El protocolo SSL no está recomendado por haber quedado obsoleto y ser vulnerable [RFC756]

247. Como hemos mencionado, el protocolo TLS permite conexiones no autenticadas o autenticadas unilateralmente (como sucede habitualmente con las conexiones HTTPS, que solo se autentican en el lado del servidor). En todo caso, son los desarrolladores quienes deben considerar si es necesaria una autenticación adicional en la capa de aplicación (caso del acceso a las cuentas bancarias, por ejemplo). En el caso de conexiones críticas, se recomienda la autenticación mediante dos factores (algo que se conoce y algo que se tiene).

248. Finalmente, el protocolo de registro utiliza el resultado de los datos de sesión establecidos para proteger la confidencialidad e integridad de las comunicaciones posteriores, por ejemplo, los datos de la aplicación o la actualización de los datos de la sesión. Los mensajes transmitidos a través del protocolo TLS se denominan registros.

249. Con relación a las diferentes versiones de los protocolos SSL y TLS disponibles, es importante señalar que ninguna de las versiones del protocolo SSL debe utilizarse, ya sea la v2 [EH95, RFC6176] o la v3 [FKK11, RFC756] (la versión v1 no fue publicada). Por su parte, TLS 1.0 es un desarrollo adicional directo de SSL 3.0 [RFC2246] por lo que tampoco debe ser utilizado. También están disponibles para el TLS las versiones 1.1 [RFC4346], 1.2 [RFC5246] y 1.3 [RFC8446], pero solo las versiones TLS 1.2 y TLS 1.3 están autorizadas para su uso.

250. Las especificaciones propias de TLS 1.3 han modificado la estructura genérica del protocolo TLS. De hecho, el TLS 1.3 solo requiere un viaje de ida y vuelta (*Round-Trip Time*) para completar el protocolo de enlace, mientras que las versiones anteriores requerían dos. Esta modificación permite al cliente transmitir datos de la aplicación desde el tercer paquete transmitido. La reducción en el número de intercambios requeridos se debe a la eliminación de ciertos mensajes presentes en versiones anteriores. Así, se han eliminado los mensajes de señalización `ChangeCipherSpec` y `ServerHelloDone`, junto con los mensajes `ClientKeyExchange` y `Server-KeyExchange` utilizados para intercambiar valores públicos del protocolo DH. Con estas modificaciones, el protocolo TLS 1.3 es como sigue:

    1. El cliente inicia una solicitud enviando un mensaje del tipo `ClientHello`, que contiene las suites criptográficas que admite y sus extensiones.

    2. El servidor responde con un `ServerHello` que contiene la suite y las extensiones seleccionadas necesarias para el intercambio de claves.

    3. El servidor envía un mensaje `EncryptedExtensions`, que contiene las otras extensiones (no necesarias para el intercambio de claves criptográficas) para esta sesión de las que no dependen los parámetros criptográficos.

    4. El servidor envía un mensaje de `Certificate`, que contiene, en particular, su clave pública en un certificado digital.

5. El servidor se autentica ante el cliente transmitiendo `CertificateVerify` que contiene datos firmados con la clave privada correspondiente a la clave pública anterior.

6. El servidor envía un `Finished`.

7. El cliente por su parte envía un `Finished`.

251. `ClientHello` contiene una lista de parámetros criptográficos que el cliente puede utilizar durante la sesión, de modo que el servidor selecciona los parámetros criptográficos para dicha sesión, después de compararlos con los que él acepta. Esta selección afecta a la forma en que se utilizarán las claves criptográficas para proteger los registros intercambiados después del protocolo de enlace, que transportan los datos de la aplicación. El propio procedimiento de negociación de claves se modifica de acuerdo con los parámetros adoptados. Los mecanismos criptográficos negociados durante el protocolo de enlace son:

   - Un mecanismo de intercambio de claves.
   - Un mecanismo de autenticación, simétrico o asimétrico, de la fase de intercambio de claves.
   - Mecanismos que garantizan la confidencialidad e integridad de los datos intercambiados después del protocolo de enlace:
     - Puede ser mediante un modo de cifrado integrado que ofrece simultáneamente cifrado e integridad.
     - O como la composición de un algoritmo de cifrado y una función resumen utilizada en modo HMAC.
   - Una función resumen empleada para la derivación de claves y en los mecanismos de autenticación asimétrica.

252. En la Tabla 4.1 se muestran las versiones recomendadas del protocolo TLS de comunicaciones.

Protocolo	Rec./Her.	Referencias	Notas
TLS 1.3	R	[RFC8446]	66
TLS 1.2	R	[RFC5246]	66

Tabla 4.1: Versiones del protocolo TLS autorizadas

Nota 66 [**Versiones de TLS**] Siempre que sea posible, se recomienda utilizar la versión 1.3 de TLS antes que la versión 1.2; no obstante, esta última también está aceptada siempre que se sigan las recomendaciones de esta guía. Así pues, no están permitidas las versiones v2 y v3 de SSL y las versiones 1.0 y 1.1 de TLS. Además, debería preferirse el uso de software que no admita ninguna de estas versiones.

253.

254. Se recomienda utilizar una suite criptográfica que ofrezca *Perfect Forward Secrecy* (PFS), es decir, secreto perfecto persistente.

255. PFS es una característica de los protocolos criptográficos que asegura que incluso si la clave privada a largo plazo se ve comprometida, las claves de sesión y las claves de sesión anteriores aún permanecen secretas. En general, esta propiedad se puede obtener mediante el uso de claves públicas efímeras.

256. **Acuerdo de clave**: Con relación a los mecanismos de acuerdo de clave, se debe garantizar la propiedad de confidencialidad persistente, lo que requiere de una suite criptográfica basada en un intercambio Diffie-Hellman con claves efímeras, esto es, claves generadas en cada nueva sesión (denotados como DHE o EC-DHE).

257. Como ya se ha mencionado, el TLS 1.3 ha evolucionado dando lugar a la posibilidad de que el cliente negocie los grupos DHE y EC-DHE. En el caso de DHE, la seguridad del intercambio está ligada, como se sabe, al orden del grupo multiplicativo. De hecho, el ataque [ABD+15] mostró la debilidad de grupos de 512 bits por lo que tampoco los grupos de 1024 son recomendables y, así, se sugiere el uso de grupos de 3072 bits o más (se aceptan grupos de 2048 bits para la protección de datos, solo hasta 2030).

258. Los grupos multiplicativos autorizados son los definidos en [RFC7919] y se listan en la Tabla 4.4 para el protocolo TLS 1.3 y en la Tabla 4.11 para el protocolo TLS 1.2. En el caso de EC-DHE, se deben utilizar grupos cuyos órdenes sean múltiplos de un número primo mayor de, al menos, 256 bits.

259. **Autenticación**: En versiones anteriores a la 1.3 del TLS, se podían utilizar diferentes mecanismos de intercambio de claves, pero no todos ellos requerían la autenticación del servidor, por lo que podían ser atacados por el ataque del MitM. Para evitar esta debilidad, es indispensable la autenticación del servidor ante el cliente mediante un mecanismo asimétrico. Para la autenticación se prefieren los métodos basados en EC-DSA. Ahora bien, debido a que el algoritmo de firma está vinculado al certificado del servidor, es posible que la autenticación basada en EC-DSA no pueda llevarse a cabo, por lo que en este caso se tolera la autenticación del servidor con el algoritmo RSA. En el proceso de autenticación del servidor se desaconsejan las alternativas anónimas o las basadas en el uso de certificados sin procesar definidos en [RFC7250]. Las funciones resumen necesarias en los procesos de autenticación se comentarán más tarde.

260. **Cifrados simétricos**: Los mecanismos de autenticación simétrica son, en general, más complejos de implementar y de mantener por lo que solo se deberían utilizar en entornos controlados.

261. El secreto compartido obtenido en el intercambio de clave permite que ambas partes (cliente y servidor) determinen una clave simétrica con la que proteger la confidencialidad de la información intercambiada que sigue a la fase de negociación. El algoritmo de cifrado a emplear se fija en la selección de una suite criptográfica y el protocolo TLS permite utilizar tanto cifradores en bloque como en flujo. Tradicionalmente, el único modo de cifrado en flujo disponible en TLS era RC4. Sin embargo, después de publicarse algunas debilidades de este sistema de cifrado [ABP+13, VP15] se ha prohibido su uso en este protocolo [RFC7465]. En lugar de emplear RC4, se ha propuesto el algoritmo de cifrado ChaCha20 [RFC7905], que se mantiene para TLS 1.3. ChaCha20 emplea claves de cifrado de 256 bits y hasta la fecha no se conoce ningún ataque a este algoritmo. En todo caso, se prefiere el uso de AES, ya sea con una clave de 128 o de 256 bits, dado que, hasta la fecha, no se conocen ataques prácticos que pongan en entredicho su seguridad.

262. **Cifrado de integridad**: Para evitar las debilidades de las versiones de TLS anteriores a la 1.2, TLS 1.2 introdujo la capacidad de utilizar modos de cifrado fuertes, proporcionando una función de cifrado combinada y una función de cálculo de patrón de integridad; por su parte, TLS 1.3 solo ofrece modos de cifrado combinados. Se han estandarizado las suites que ofrecen los modos de funcionamiento GCM y CCM, también se permite el modo ChaCha20_Poly1305 [RFC8439]. Los modos combinados de GCM y ChaCha20_Poly1305 requieren especial atención al administrar las claves de un solo uso (*nonces*). En estos dos modos, el cifrado de cada registro requiere el uso de un *nonce* único durante la sesión y si no se garantiza la unicidad del mismo, la confidencialidad y la integridad de los datos intercambiados puede quedar comprometida. En el caso del TLS 1.2 no se especifica cómo se debe generar este *nonce*; mientras que en el TLS 1.3 la construcción de estos *nonces* sí se ha incorporado a sus especificaciones.

263. **Funciones resumen**: Como ya se ha mencionado, el protocolo TLS necesita una función resumen para la derivación de la clave secreta, para calcular la firma durante la autenticación asimétrica y para determinar los patrones de integridad cuando se utiliza el modo de cifrado CBC. Las versiones TLS 1.2 y TLS 1.3 usan SHA-256 o SHA-384 dependiendo del paquete criptográfico seleccionado. En el caso de que no se use un modo de cifrado fuerte, el patrón de integridad se calcula mediante el modo HMAC. A pesar de que en las versiones de TLS se permite el uso de diferentes funciones resumen, solo las funciones de la familia SHA-2 y SHA-3 deben ser utilizadas.

4.1.1. TLS Versión 1.3

264. A continuación se presentan las tablas que contienen los diferentes conjuntos de mecanismos criptográficos autorizados para la versión 1.3 del protocolo TLS.

265. La Tabla 4.2 presenta la suite de cifradores para la versión 1.3 de TLS. La convención que se sigue para esta tabla es TLS_ENC_Long_Mod_Hash, siendo ENC el sistema de cifrado, Long es la longitud de la clave considerada, Mod es el modo de operación del cifrado y Hash hace referencia a la función resumen considerada.

Código	Suites criptográficas	Rec./Her.	Referencias	Notas
0x1302	TLS_AES_256_GCM_SHA384	R	[RFC8446]	
0x1301	TLS_AES_128_GCM_SHA256	R	[RFC8446]	
0x1304	TLS_AES_128_CCM_SHA256	R	[RFC8446]	
0x1303	TLS_CHACHA20_POLY1305_SHA256	R	[RFC7905]	

Tabla 4.2: Suites criptográficas autorizadas para el protocolo TLS 1.3

266. Además del acuerdo de clave de Diffie-Hellman sobre cuerpos finitos o curvas elípticas, TLS 1.3 ofrece modos de protocolo de enlace adicionales utilizando claves precompartidas o PSK (*Pre-Shared Key*). En este contexto, las PSK se refieren a claves que se proporcionan fuera de banda o al material de claves que se ha establecido en una sesión anterior a través del mecanismo de ticket de sesión. En la Tabla 4.3 se presentan los modos PSK recomendados para TLS 1.3.

Código	Modo PSK	Rec./Her.	Referencias	Notas
0x0000	psk_ke	H[2026]	[RFC8446]	67, 68
0x0001	psk_dhe_ke	R	[RFC8446]	68

Tabla 4.3: Modos de clave precompartida recomendados para el protocolo TLS 1.3

267. Nota 67 **[Modo psk_ke de PSK]** El modo PSK psk_ke no ofrece secreto perfecto persistente. Este modo solo debe usarse en aplicaciones especiales después de consultar a un experto.

Nota 68 **[Datos 0-RTT]** El protocolo TLS 1.3 ofrece una opción para incluir datos de aplicación ya en el primer mensaje de un protocolo de enlace PSK, son los datos llamados de tiempo cero de ida y vuelta o datos 0-RTT (*zero Round-Trip Time data*). Estos datos no están protegidos contra ataques de reproducción por lo que no se recomienda enviar o aceptar datos de este tipo.

268.

269. Existen algunas extensiones para el protocolo TLS 1.3 que se mostrarán a continuación y que tienen que ver con los grupos que se pueden utilizar, los algoritmos de firma, etc.

270. En el TLS 1.3, el cliente y el servidor pueden usar la extensión "supported_groups" para informarse mutuamente sobre los grupos Diffie-Hellman que desean usar para (EC)DHE. En la Tabla 4.4 se listan los grupos Diffie-Hellman recomendados.

Código	Grupo DH	Rec./Her.	Referencias	Notas
0x0100	ffdhe2048	H[2025]	[RFC7919]	
0x0101	ffdhe3072	R	[RFC7919]	
0x0102	ffdhe4096	R	[RFC7919]	
0x0017	secp256r1 (P-256)	R	[RFC8422]	
0x0018	secp384r1 (P-384)	R	[RFC8422]	
0x0019	secp521r1 (P-521)	R	[RFC8422]	
0x001F	brainpoolP256r1tls13	R	[RFC8734]	
0x0020	brainpoolP384r1tls13	R	[RFC8734]	
0x0021	brainpoolP512r1tls13	R	[RFC8734]	

Tabla 4.4: Grupos de Diffie-Hellman recomendados para el protocolo TLS 1.3

271. En TLS 1.3, el cliente y el servidor pueden usar las extensiones "signature_algorithms" y "signature_algorithms_cert" para informarse mutuamente sobre los algoritmos de firma que desean usar para la autenticación basada en certificados. La extensión "signature_algorithms" hace referencia a firmas que son generadas por el cliente o servidor para su mensaje `CertificateVerify`; mientras que la extensión "signature_algorithms_cert" se refiere a firmas en certificados. En la Tabla 4.5 se listan los algoritmos de firma recomendados para la extensión "signature_algorithms" y en la Tabla 4.6 se presentan los algoritmos de firma recomendados para la extensión "signature_algorithms_cert".

Código	Algoritmo de firma	Rec./Her.	Referencias	Notas
0x0804	rsa_pss_rsae_sha256	R	[RFC8446]	
0x0805	rsa_pss_rsae_sha384	R	[RFC8446]	
0x0806	rsa_pss_rsae_sha512	R	[RFC8446]	
0x0809	rsa_pss_pss_sha256	R	[RFC8446]	
0x080A	rsa_pss_pss_sha384	R	[RFC8446]	
0x080B	rsa_pss_pss_sha512	R	[RFC8446]	
0x0403	ecdsa_secp256r1_sha256	R	[RFC8446]	
0x0503	ecdsa_secp384r1_sha384	R	[RFC8446]	
0x0603	ecdsa_secp521r1_sha512	R	[RFC8446]	
0x081A	ecdsa_brainpoolP256r1tls13_sha256	R	[RFC8734]	
0x081B	ecdsa_brainpoolP384r1tls13_sha384	R	[RFC8734]	
0x081C	ecdsa_brainpoolP512r1tls13_sha512	R	[RFC8734]	

Tabla 4.5: Algoritmos de firma (cliente/servidor) recomendados para el protocolo TLS 1.3

Código	Funciones resumen	Rec./Her.	Referencias	Notas
0x0401	rsa_pkcs1_sha256	H[2025]	[RFC8446]	69
0x0501	rsa_pkcs1_sha384	H[2025]	[RFC8446]	69
0x0601	rsa_pkcs1_sha512	H[2025]	[RFC8446]	69
0x0804	rsa_pss_rsae_sha256	R	[RFC8446]	
0x0805	rsa_pss_rsae_sha384	R	[RFC8446]	
0x0806	rsa_pss_rsae_sha512	R	[RFC8446]	
0x0809	rsa_pss_pss_sha256	R	[RFC8446]	
0x080A	rsa_pss_pss_sha384	R	[RFC8446]	
0x080B	rsa_pss_pss_sha512	R	[RFC8446]	
0x0403	ecdsa_secp256r1_sha256	R	[RFC8446]	
0x0503	ecdsa_secp384r1_sha384	R	[RFC8446]	
0x0603	ecdsa_secp521r1_sha512	R	[RFC8446]	
0x081A	ecdsa_brainpoolP256r1tls13_sha256	R	[RFC8734]	
0x081B	ecdsa_brainpoolP384r1tls13_sha384	R	[RFC8734]	
0x081C	ecdsa_brainpoolP512r1tls13_sha512	R	[RFC8734]	

Tabla 4.6: Algoritmos de firma (en certificados) recomendados para el protocolo TLS 1.3

272. Nota 69 **[Firma rsa_pkcs1_*]** El uso de los algoritmos de firma rsa_pkcs1_* (códigos 0x0401, 0x0501 y 0x0601) solo se admite hasta 2025, porque utilizan el esquema de relleno PKCS # 1 v1.5.

4.1.2. TLS Versión 1.2

273. En la versión 1.2 de TLS, los mecanismos criptográficos de una conexión se definen mediante una suite criptográfica que especifica un mecanismo de acuerdo de claves (con autenticación) para el protocolo de enlace, un algoritmo de cifrado autenticado para el protocolo de registro y una función resumen para el proceso de derivación de claves. Dependiendo de la suite criptográfica, también se debe especificar un grupo para el protocolo de Diffie-Hellman, ya sea en un subgrupo de un cuerpo finito o en una curva elíptica sobre un cuerpo finito, y un algoritmo de firma para el acuerdo de claves.

274. Para estas suites criptográficas se suele utilizar la siguiente notación: TLS_AKE_WITH_ENC_Hash, donde AKE denota un mecanismo de acuerdo de clave (con autenticación), ENC es un algoritmo de cifrado que incluye la longitud de su clave y el modo de operación, y Hash hace referencia a la función resumen. La función resumen se utiliza para un HMAC que se emplea en una PRF, la cual es usada para la derivación de claves. En el caso de que el sistema de cifrado considerado utilice el modo de operación CCM, no se indica ninguna función resumen y se da por hecho que para la PRF usa SHA-256. Si ENC no fuera un algoritmo AEAD, entonces el HMAC también se utiliza para la protección de la integridad en el protocolo de registro.

275. Se muestran a continuación las tablas que contienen las diferentes suites criptográficas autorizadas para el protocolo TLS 1.2.

276. En las Tablas 4.7 y 4.8 se muestran las suites criptográficas recomendadas para la versión 1.2 de TLS en los casos en los que el servidor disponga de un certificado con la clave pública ECDSA o RSA, respectivamente.

Código	Suites criptográficas	Rec./He	Referencias
0xC02C	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	R	[RFC5289]
0xC02B	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	R	[RFC5289]
0xC0AD	TLS_ECDHE_ECDSA_WITH_AES_256_CCM	R	[RFC7251]
0xC0AC	TLS_ECDHE_ECDSA_WITH_AES_128_CCM	R	[RFC7251]
0xCCA9	TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256	R	[RFC7905]

Tabla 4.7: Suites criptográficas recomendadas para TLS 1.2 con un servidor que disponga de un certificado con la clave pública EC-DSA

Código	Suites criptográficas	Rec./Her.	Referencias
0xC030	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	H	[RFC5289]
0xC02F	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	H	[RFC5289]
0xCCA8	TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256	H	[RFC7905]

Tabla 4.8: Suites criptográficas heredadas para TLS 1.2 con un servidor que disponga de un certificado con la clave pública RSA

277. Cuando una de las dos partes en una comunicación no es la dominante, no siempre es posible negociar una sesión TLS con una de las suites criptográficas anteriores. Si se ha identificado una gran necesidad de compatibilidad, se pueden adoptar otras suites, con detrimento de la seguridad en las comunicaciones. En este caso, es necesario evaluar el perfil de los servidores o de los clientes interesados y adoptar solo las suites que se consideren esenciales para llevar a cabo las funciones de la aplicación consideradas.

278. La Tabla 4.9 lista las suites criptográficas recomendadas para uso general con TLS 1.2 cuando no hay soporte ECC o modo de cifrado autenticado.

Código	Suites criptográficas	Rec./Her.	Referencias	Notas
0xC024	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	H[2025]	[RFC5289]	70
0xC023	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	H[2025]	[RFC5289]	70
0xC028	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	H[2025]	[RFC5289]	70
0xC027	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	H[2025]	[RFC5289]	70
0x009F	TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	H	[RFC5288]	71
0x009E	TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	H	[RFC5288]	71
0xC09F	TLS_DHE_RSA_WITH_AES_256_CCM	H	[RFC6655]	71
0xC09E	TLS_DHE_RSA_WITH_AES_128_CCM	H	[RFC6655]	71
0x006B	TLS_DHE_RSA_WITH_AES_256_CBC_SHA256	H[2025]	[RFC5246]	70,71
0x0067	TLS_DHE_RSA_WITH_AES_128_CBC_SHA256	H[2025]	[RFC5246]	70,71
0x009D	TLS_RSA_WITH_AES_256_GCM_SHA384	H		72
0x009C	TLS_RSA_WITH_AES_128_GCM_SHA256	H		72
0xC09D	TLS_RSA_WITH_AES_256_CCM	H		72
0xC09C	TLS_RSA_WITH_AES_128_CCM	H		72
0x003D	TLS_RSA_WITH_AES_256_CBC_SHA256	H[2025]		70,72
0x003C	TLS_RSA_WITH_AES_128_CBC_SHA256	H[2025]		70,72
0xCCAA	TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256	H	[RFC7905]	
0x006A	TLS_DHE_DSS_WITH_AES_256_CBC_SHA256	H	[RFC5246]	
0x00A3	TLS_DHE_DSS_WITH_AES_256_GCM_SHA384	H	[RFC5288]	

Tabla 4.9: Suites criptográficas recomendadas para TLS 1.2 cuando no hay soporte ECC o modo de cifrado autenticado

279. Nota 70 **[TLS Cifrado y luego MAC]** Las suites criptográficas TLS cuyos mecanismos de cifrado se basan en CBC deben utilizarse junto con la extensión encrypt_then_mac.

280. Nota 71 **[TLS con DHE]** Cuando se utiliza el intercambio de claves DHE en TLS, el servidor impone los parámetros del grupo al cliente y como la validación de estos parámetros por parte del cliente puede ser problemática, se debe preferir el uso de EC-DHE para el intercambio de claves, dado que en este caso, los parámetros de grupo se negocian en el protocolo de enlace.

281. Nota 72 **[TLS con RSA]** El intercambio de clave con RSA no ofrece PFS.

282. Si los datos adicionales que se han intercambiado de antemano se van a incorporar en el acuerdo de clave, se pueden utilizar suites criptográficas con una PSK. En general, se recomienda utilizar suites criptográficas para los que se incorporan al acuerdo de claves más claves efímeras o números aleatorios previamente intercambiados, además de la clave precompartida. En el caso del protocolo TLS 1.2 cuando se utilizan suites criptográficas con una clave previamente compartida no se recomienda el uso de suites criptográficas de tipo TLS_PSK_*, es decir, sin claves efímeras o números aleatorios adicionales, porque la seguridad de la conexión se basa únicamente en la entropía y la confidencialidad de las claves previamente compartidas para estos conjuntos de cifrado.

283. En la Tabla 4.10 se incluyen las suites criptográficas con PSK que se recomiendan.

Código	Suites criptográficas	Rec./Her.	Referencias	Notas
0xC037	TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256	R	[RFC5489]	
0xC038	TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA384	R	[RFC5489]	
0xD001	TLS_ECDHE_PSK_WITH_AES_128_GCM_SHA256	R	[RFC8442]	
0xD002	TLS_ECDHE_PSK_WITH_AES_256_GCM_SHA384	R	[RFC8442]	
0xD005	TLS_ECDHE_PSK_WITH_AES_128_CCM_SHA256	R	[RFC8442]	
0xCCAD	TLS_DHE_PSK_WITH_CHACHA20_POLY1305_SHA256	R	[RFC7905]	
0x00B2	TLS_DHE_PSK_WITH_AES_128_CBC_SHA256	R	[RFC5487]	
0x00B3	TLS_DHE_PSK_WITH_AES_256_CBC_SHA384	R	[RFC5487]	
0x00AA	TLS_DHE_PSK_WITH_AES_128_GCM_SHA256	R	[RFC5487]	
0x00AB	TLS_DHE_PSK_WITH_AES_256_GCM_SHA384	R	[RFC5487]	
0xC0A6	TLS_DHE_PSK_WITH_AES_128_CCM	R	[RFC6655]	
0xC0A7	TLS_DHE_PSK_WITH_AES_256_CCM	R	[RFC6655]	

Tabla 4.10: Suites criptográficas recomendadas para TLS 1.2 con clave precompartida

284. En cuanto a las extensiones para TLS 1.2, a continuación se presentan las recomendadas para esta versión del protocolo.

285. Se recomienda el uso de la extensión "supported_groups" para los conjuntos de cifrado TLS_DHE_* tan pronto como estén disponibles las implementaciones correspondientes. Así, en la Tabla 4.11 se listan los grupos Diffie-Hellman recomendados.

Código	Grupo DH	Rec./Her.	Referencias
0x0100	ffdhe2048	H[2025]	[RFC7919]
0x0101	ffdhe3072	R	[RFC7919]
0x0102	ffdhe4096	R	[RFC7919]
0x0017	secp256r1 (P-256)	R	[RFC8422]
0x0018	secp384r1 (P-384)	R	[RFC8422]
0x001A	brainpoolP256r1	R	[RFC7027]
0x001B	brainpoolP384r1	R	[RFC7027]
0x001C	brainpoolP512r1	R	[RFC7027]

Tabla 4.11: Grupos de Diffie-Hellman recomendados para el protocolo TLS 1.2

286. En TLS 1.2, el cliente puede usar la extensión "signature_algorithms" [RFC5246] para informar al servidor sobre los algoritmos de firma que desea usar para acuerdos de clave y certificados. El algoritmo debe especificarse como una combinación de un algoritmo de firma y una función resumen. En la Tabla 4.12 se listan los algoritmos de firma recomendados y en la Tabla 4.13 las funciones resumen para tales algoritmos de firma.

Código	Algoritmo de firma	Rec./Her.	Referencias
0x0001	rsa	H[2025]	[RFC5246]
0x0002	dsa	R	[RFC5246]
0x0003	ecdsa	R	[RFC5246]

Tabla 4.12: Algoritmos de firma recomendados para el protocolo TLS 1.2

Código	Funciones resumen	Rec./Her.	Referencias
0x0004	sha256	R	[RFC5246]
0x0005	sha384	R	[RFC5246]
0x0006	sha512	R	[RFC5246]

Tabla 4.13: Funciones resumen para el protocolo TLS 1.2

4.2. SSH

287. SSH (*Secure Shell*) es un protocolo criptográfico de seguridad [RFC4251] diseñado originalmente para sustituir a los protocolos de conexión remota inseguros como, por ejemplo, Telnet (la página oficial de este protocolo es https://www.ssh.com/academy/ssh). Este protocolo se puede utilizar para establecer un canal seguro dentro de una red insegura. Las aplicaciones más comunes del protocolo SSH son

iniciar sesión en un sistema remoto (inicio de sesión de línea de comando remoto) y ejecutar comandos o ejecutar aplicaciones en sistemas remotos [TR-02102-].

288. La primera versión del protocolo SSH se conoce como SSH-1 [Ylö96], pero su uso no está recomendado. La versión que sí está recomendada es la versión 2, denotada por SSH-2 [RFC4251].

289. El protocolo SSH está formado por tres subprotocolos: Protocolo de capa de transporte, Protocolo de autenticación de usuario y Protocolo de conexión. El Protocolo de la capa de transporte [RFC4253] permite la autenticación del servidor, el cifrado, la protección de la integridad y, opcionalmente, la compresión de datos. Se basa en el protocolo TCP/IP. El Protocolo de autenticación de usuario [RFC4252] se utiliza para autenticar al usuario en el servidor y se basa en el Protocolo de la capa de transporte. Finalmente, el Protocolo de conexión [RFC4254] es el responsable de crear y administrar canales lógicos dentro del túnel cifrado y se basa en el Protocolo de autenticación de usuarios.

4.2.1. Acuerdo de Clave

290. Cuando se establece una conexión SSH se intercambian las claves con el fin de crear e intercambiar claves de sesión compartidas para la autenticación y el cifrado. El mecanismo de acuerdo de clave del protocolo SSH se basa en el de Diffie-Hellman. De hecho, hay varios grupos recomendados, todos ellos con la función SHA512. Los números primos y el generador de cada grupo están publicados en [RFC3526]. En el caso del acuerdo de clave con curvas elípticas, se emplea el mecanismo ECDH [RFC5656]. En la Tabla 4.14 se presentan las versiones recomendadas del protocolo SSH.

Protocolo	Rec./Her.	Referencias	Notas
DH-exchange-SHA256	R	[RFC4419, §4.2]	73, 74
DH-grupo15-SHA512	R	[RFC3526, §4]	74
DH-grupo16-SHA512	R	[RFC3526, §5]	74
DH-grupo17-SHA512	R	[RFC3526, §6]	74
DH-grupo18-SHA512	R	[RFC3526, §7]	74
ECDH-SHA2-*	R	[RFC5656, §6.3]	74, 75

Tabla 4.14: Versiones de acuerdos de clave para el protocolo SSH autorizadas

291. Nota 73 [**Diffie-Hellman-group-exchange-SHA256**] El tamaño del número primo a emplear, p, debe ser de, al menos, 3000 bits. Además, el orden del generador debe tener un tamaño de 2^{250}, como mínimo. Finalmente, p debe ser un primo seguro. Recuérdese que un primo seguro, p, es un primo de la forma $p = 1 + 2q$, siendo q otro primo. Por tanto, el orden del generador g solo puede ser un divisor de $p - 1 = 2 \cdot q$, esto es, o es 2 o q. Esto significa que el tamaño de q debe ser mucho mayor que el mínimo recomendado de 2^{250} para el orden de g.

292. Nota 74 [**Renovación de la clave**] Con el fin de más dificultar el ataque contra las claves de sesión, se recomienda renovar la clave utilizada en una conexión después de un cierto período de tiempo o una cierta cantidad de datos transmitidos. Con SSH este proceso lo pueden iniciar tanto el cliente como el servidor, para lo que es suficiente con enviar el mensaje `SSH_MSG_KEXINIT`. Así pues, siguiendo la recomendación de [RFC4253, Cap. 9], las claves de sesión se deben renovar después de una hora o después de que se haya transmitido un gigabyte (lo que ocurra primero).

293. Nota 75 [**ECDH-SHA2-***] El intercambio de claves ECDH se define mediante una familia de nombres de métodos. Cada nombre de método es la concatenación de la cadena ECDH-SHA2- con el parámetro de dominio de la correspondiente curva elíptica. Las curvas elípticas acordadas son las siguientes: nistp256, nistp384 y nistp521 [SP800-186], también denominadas respectivamente secp256r1, secp384r1 y secp521r1 [SEC10].

4.2.2. Cifrado

294. Durante el intercambio de claves, el cliente y el servidor acuerdan un algoritmo de cifrado y una clave de cifrado compartida. Los algoritmos de cifrado recomendados se muestran en la Tabla 4.15.

Protocolo	Rec./Her.	Referencias	Notas
AEAD_AES_128_GCM	R	[RFC5647, §6.1]	76, 77
AEAD_AES_256_GCM	R	[RFC5647, §6.2]	76, 77
AES128-CTR	R	[RFC4344, §4]	77
AES192-CTR	R	[RFC4344, §4]	77
AES256-CTR	R	[RFC4344, §4]	77

Tabla 4.15: Algoritmos de cifrado autorizados para el protocolo

295. Nota 76 **[Protección MAC]** Los algoritmos AEAD_AES_128_GCM y AEAD_AES_128_GCM ya incluyen la protección MAC en el modo GCM por lo que se deberían utilizar con preferencia a los restantes.

296. Nota 77 **[Declaración de seguridad]** En la medida de lo posible, deben utilizarse los algoritmos AEAD_AES_128_GCM y AEAD_AES_128_GCM puesto que existen declaraciones de seguridad comprobables para el modo GCM con respecto al objetivo de seguridad del cifrado autenticado. Si se escoge cifrado no autenticado (i.e., AES-CTR) será obligatorio utilizar alguna de las opciones recomendadas para proteger la integridad y autenticidad de los mensajes (i.e., HMAC-SHA2).

4.2.3. Integridad y Autenticidad en Origen

297. La integridad y la autenticidad en origen se pueden llevar a cabo mediante la protección por MAC, para lo cual se recomiendan las funciones que se muestran en la Tabla 4.16.

Función	Rec./Her.	Referencias	Notas
HMAC-SHA1	H[2030]	[RFC4253, §6.4]	
HMAC-SHA2-256	R	[RFC6668, §2]	
HMAC-SHA2-512	R	[RFC6668, §2]	
AEAD_AES_128_GCM	R	[RFC5647]	
AEAD_AES_256_GCM	R	[RFC5647]	

Tabla 4.16: Funciones MAC autorizadas para el protocolo SSH

4.2.4. Autenticación del Servidor y del Cliente

298. La autenticación de servidor SSH se lleva a cabo mediante criptografía asimétrica usando claves públicas o certificados. En la Tabla 4.17 se presentan los métodos recomendados para la autenticación del servidor en el protocolo SSH.

Método	Rec./Her.	Referencias	Notas
ECDSA-SHA2-*	R	[RFC5656, §3]	78
x509v3-ECDSA-SHA2-*	R	[RFC6187, §3.4]	79

Tabla 4.17: Métodos de autenticación del servidor autorizados para el protocolo SSH

299. Nota 78 [**ECDSA-SHA2-***] El protocolo ECDSA se define mediante una familia de nombres de métodos. Cada nombre de método es la concatenación de la cadena ECDSA-SHA2- con el parámetro de dominio de la correspondiente curva elíptica. Las curvas elípticas acordadas son las siguientes: nistp256, nistp384 y nistp521 [SP800-186], también denominadas respectivamente secp256r1, secp384r1 y secp521r1 [SEC10].

300. Nota 79 [**x509v3-ECDSA-***] El protocolo x509v3-ECDSA se define mediante una familia de nombres de métodos. Cada nombre de método es la concatenación de la cadena x509v3-ECDSA-SHA2- con el parámetro de dominio de la correspondiente curva elíptica. Las curvas elípticas acordadas son las siguientes: nistp256, nistp384 y nistp521 [SP800-186], también denominadas respectivamente secp256r1, secp384r1 y secp521r1 [SEC10].

4.3. IPSEC con IKEV2

301. En esta sección se tratará la seguridad del protocolo de Internet o IPsec (*Internet Protocol Security*) [RFC8221] y el protocolo de intercambio de claves de Internet o IKE (*Internet Key Exchange*), cuya versión 2 se denota por IKEv2 [RFC7296]. En esta guía no se considera la versión 1 de este protocolo (IKEv1).

302. IPsec es un estándar que proporciona seguridad a nivel de capa de red del Protocolo de Internet o IP (*Internet Protocol*) en la pila del protocolo TCP/IP (*Transmission Control Protocol/Internet Protocol*). A diferencia de los protocolos TLS (ver §4.1)

y SSH (ver §4.2), IPsec proporciona seguridad en las capas superiores, como la de aplicación.

303. El uso más importante de IPsec es el de crear redes privadas virtuales o VPN (*Virtual Private Network*), esto es, establecer canales de comunicación seguros mediante redes IP que no son seguras.

304. IPsec se puede desplegar en el modo túnel y en el de transporte. En el modo túnel, el paquete IP se protege de modo criptográfico al completo y en modo transporte, la cabecera del paquete IP original se conserva y se le añaden algunos campos de seguridad.

305. Nota 80 [**Modo túnel-Modo transporte**] Se puede afirmar que el modo túnel ofrece más seguridad que el modo transporte, por lo que se deberá usar aquella configuración antes que esta. El modo transporte está justificado solo si el tamaño del paquete presenta problemas debido a las restricciones que imponga la red.

306. En la Tabla 4.18 se muestran las versiones del protocolo IPsec autorizadas.

Protocolo	Rec./Her.	Referencias	Notas
IPsec	R	[RFC8221]	81
IKEv2	R	[RFC7296], [RFC8247]	
ESP	R	[RFC4303]	81

Tabla 4.18: Versiones autorizadas del protocolo IPsec

307. Nota 81 [**Ataques a IPsec**] Existen ataques conocidos cuando se implementa IPsec en cualquier configuración *MAC-then-Encrypt* (como, por ejemplo, si se usa AH en modo transporte antes de un ESP sólo cifrado en modo túnel). Sin embargo, no se conocen ataques a IPsec si se usa ESP sólo cifrado seguido de AH o ESP con autenticidad e integridad sin que le siga otro AH. Por lo tanto, se recomienda el uso de ESP siempre con las opciones de protección de integridad y autenticidad, además de confidencialidad.

4.3.1. Acuerdo de Claves

308. El mecanismo de acuerdo de clave empleado en el protocolo IKEv2 se basa en el de Diffie-Hellman. Se consideran varios grupos de tipo Diffie-Hellman para ser empleados en IKEv2, ya sean de exponenciación modular o MODP [RFC3526],

basados en curva elípticas modulo un primo, ECP (curva elípticas modulo un primo) [RFC5903] o curvas Brainpool [RFC8031].

309. En la Tabla 4.19 se muestran los los grupos de DH y ECDH acordados.

Grupo (EC)DH	Rec./Her.	Referencias	Notas
3072-bit MODP	R	[RFC3526]	
4096-bit MODP	R	[RFC3526]	
6144-bit MODP	R	[RFC3526]	
8192-bit MODP	R	[RFC3526]	
256-bit aleatorio ECP	R	[RFC5903]	
384-bit aleatorio ECP	R	[RFC5903]	
521-bit aleatorio ECP	R	[RFC5903]	
brainpoolP256r1	R	[RFC6954]	
brainpoolP384r1	R	[RFC6954]	
brainpoolP512r1	R	[RFC6954]	
x25519	R	[RFC8031]	
x448	R	[RFC8031]	

Tabla 4.19: Grupos de tipo DH y ECDH acordados por IKEv2

4.3.2. Cifrado

310. Las propuestas para los esquemas de cifrado IKEv2 y ESP pueden incluir tanto esquemas de cifrado clásico como esquemas AEAD. Atendiendo a las recomendaciones señaladas en las subsecciones 2.2.1 y 2.2.5, se acuerdan los esquemas de cifrado o esquemas de cifrado autenticado que se incluyen en las Tablas 2.5 y 2.9.

4.3.3. Integridad y Autenticación

311. Los protocolos IKEv2 y ESP utilizan mecanismos MAC para la verificación de la integridad y la autenticación de origen. Los mecanismos MAC autorizados para uso recomendado son los basados en esquemas AES-CMAC, AES-GMAC y HMAC-SHA2. Por su parte, los esquemas HMAC-SHA2, cuando se utilizan en IPsec como mecanismos de integridad y autenticidad, la longitud de la clave sera fija en función del tamaño del valor hash de salida, y se realiza un truncado de la salida tal y como se comenta en las notas de la Tabla 4.20.

Esquema (H)MAC	Rec./Her.	Referencias	Notas
HMAC-SHA2-256_128	R	[RFC4868]	82
HMAC-SHA2-384_192	R	[RFC4868]	82
HMAC-SHA2-512_256	R	[RFC4868]	82
AES-CMAC-96	R	[RFC4494]	
AES-GMAC	R	[RFC4543]	

Tabla 4.20: Esquemas MAC y HMAC autorizados para IKEv2 y ESP

312. Nota 82 **[HMAC-SHA-XXX-YYY]** Cada uno de estos esquemas utiliza una longitud de clave fija de XXX bits, truncando la salida a YYY bits.

4.3.4. Funciones Pseudo-aleatorias

313. Como ya se ha mencionado, las claves para el cifrado y la autenticación de mensajes se derivan del secreto compartido obtenido con el intercambio Diffie-Hellman, utilizando la PRF negociada. Estas funciones para IKEv2 son similares a los mecanismos MAC señalados en la Tabla 4.20, con la excepción de que se anula la restricción de claves de tamaño fijo y se elimina el truncado, debido al tamaño de la salida de la función. En la Tabla 4.21 se muestran las PRF autorizadas.

Esquema (H)MAC	Rec./Her.	Referencias	Notas
HMAC-SHA2-256	R	[RFC4868]	
HMAC-SHA2-384	R	[RFC4868]	
HMAC-SHA2-512	R	[RFC4868]	
AES128-CMAC	R	[RFC4615]	

Tabla 4.21: Funciones pseudo-aleatorias autorizadas para IKEv2

4.3.5. Mecanismos de Autenticación

314. El mecanismo de autenticación para IKEv2 es la firma electrónica, recomendándose el uso de certificados X.509. Los esquemas de firma autorizados para uso con IKEv2 se basan en ECDSA y RSA y se muestran en la Tabla 4.22.

Esquema de firma	Rec./Her.	Referencias	Notas
RSA (RSASSA-PSS)	R	[RFC4055]	83
ECDSA-SHA-256 y P-256	R	[RFC4754]	84
ECDSA-SHA-384 y P-384	R	[RFC4754]	84
ECDSA-SHA-512 y P-521	R	[RFC4754]	84
ECDSA-256-BrainpoolP256r1	R	[RFC7427]	84
ECDSA-384-BrainpoolP384r1	R	[RFC7427]	84
ECDSA-512-BrainpoolP512r1	R	[RFC7427]	84
ECGDSA-256-BrainpoolP256r1	R	[RFC7427]	
ECGDSA-384-BrainpoolP384r1	R	[RFC7427]	
ECGDSA-512-BrainpoolP512r1	R	[RFC7427]	

Tabla 4.22: Esquemas de firma acordados para IKEv2

315. Nota 83 (**RSASSA-PSS**) Este esquema solo debe utilizarse con PSS [RFC8017, §8 y §9.1] y con una función de la familia SHA-2.

316. Nota 84 (**ECDSA-***) Cuando se elabora una firma de tipo ECDSA, debe tenerse en cuenta que el *nonce* utilizado en el protocolo se selecciona aleatoriamente y se distribuye uniformemente dentro del intervalo $[1, q-1]$, siendo q el orden del punto base de la curva elíptica considerada.

5. Generadores de Números Aleatorios

317. Es bien sabido que muchas aplicaciones criptográficas requieren números aleatorios, como por ejemplo la generación de claves, ya sea para ser utilizadas un largo periodo de tiempo (asimétricas), uno corto (simétricas), para una sola vez (claves efímeras y *nonces*); o para generar determinados parámetros del sistema (desafíos, etc.). Por ello, es básico establecer los tipos y propiedades que deben verificar los generadores de números aleatorios.

5.1. Generadores de Números Aleatorios

318. El objetivo a la hora de generar números aleatorios suele ser el de producir bits (0 y 1) de modo que se distribuyan uniformemente en el conjunto $\{0,1\}^n$ (ver [ANS20a], [ACM1.3], [ANS21], [BSI22]). Esta generación de bits puede transformarse de modo inmediato a la generación de números. Además, la mayoría de las aplicaciones criptográficas precisa de determinado grado de imprevisibilidad y que los bits o números generados sean secretos. De hecho, a los generadores que se puedan usar se les exige la propiedad de que si un adversario llegara a conocer largas subsecuencias de los números aleatorios generados, no debería poder determinar predecesores o sucesores de la subsecuencia conocida. Dicho de otro modo, conocida determinada subsecuencia de números, la probabilidad de conocer el siguiente o el anterior número de la subsecuencia no debería ser mayor de $1/2$. Así pues, en las aplicaciones criptográficas es fundamental utilizar generadores de números aleatorios fuertes y seguros.

319. Dos fuentes de gran interés para la elección y estudio de los generadores de números aleatorios o RNG (*Random Number Generator*)[9] pertenecen al esquema alemán, conocidas como AIS 31 [BSI13b] para el caso de los generadores físicos de números aleatorios o PTRNG (*Physical True Random Number Generator*) y AIS 20 [BSI13a], para los generadores deterministas de números aleatorios o DRNG (*Deterministic Random Number Generator*). Para ambas fuentes, es de interés el anexo de Killman y Schindler [KS11], que define la clases de funcionalidad para generadores físicos de números aleatorios PTG.1–PTG.3, para generadores deterministas de números aleatorios DRG.1–DRG.4 y para generadores de números aleatorios no físicos ni deterministas o NPTRNG (*Non Physical True Random Number Generator*) NTG.1.

[9] Cuando se habla de generadores de bits aleatorios en lugar de números aleatorios, estos se denotan por RBG (*Random Bit Generator*). Suele ser indiferente hacer referencia a un tipo o a otro de generador puesto que ambas generaciones pueden considerarse equivalentes: todo número se puede transformar en una colección de bits y viceversa.

5.2. Generadores Físicos de Números Aleatorios

320. Los generadores físicos de números aleatorios utilizan hardware dedicado (v.gr. un circuito electrónico) como generador de números realmente aleatorios o TRNG (*True Random Number Generator*), es decir, números aleatorios impredecibles. En general se hace uso del comportamiento impredecible del hardware empleado, de modo que a la postre, la entropía de la señal se debe a un nivel físico o a las influencias ambientales dentro del sistema empleado. En muchos casos, es preciso utilizar un postprocesamiento determinista de los datos del ruido digitalizados tal como se obtienen de la fuente (*raw noise data*) con el fin de eliminar cualquier sesgo o dependencia.

321. Así pues, una fuente realmente aleatoria de números puede entenderse como un procedimiento probabilístico que proporciona bits aleatorios. En general, es muy difícil evaluar la calidad de la salida de una fuente aleatoria y se suelen emplear dos aproximaciones: 1) Mediante pruebas estadísticas a la salida de la fuente y 2) Modelando el proceso probabilístico de la fuente empleada.

322. En el primer caso (pruebas estadísticas a la salida de la fuente), el enfoque se considera es el de caja negra, es decir, no se necesita ningún conocimiento sobre la fuente para realizar las pruebas correspondientes. Este caso tiene dos inconvenientes. El primero, es que las pruebas estadísticas son genéricas y solo se pueden usar para detectar deficiencias de la fuente cuando se compara con una fuente aleatoria ideal. El segundo inconveniente es que las pruebas realizadas no proporcionan ninguna garantía acerca de la distribución de la salida de la fuente aleatoria. En cualquier caso, las pruebas estadísticas son útiles para detectar fallos de la fuente aleatoria, dicho de otro modo, pasar estas pruebas no es una garantía de calidad aleatoria, pero no pasarlas es síntoma inequívoco de mala calidad.

323. En la segunda aproximación (modelado del proceso probabilístico de la fuente) se necesita un gran conocimiento del diseño de fuente aleatoria y, en este caso, se intenta evaluar la calidad de la fuente a partir del estudio de un modelo teórico de la misma. Este enfoque proporciona una mayor seguridad, pero como contrapartida, requiere un alto nivel de experiencia en dominios como la estadística y la física.

324. En términos generales, los generadores de números aleatorios compatibles con PTG.2 o con PTG.3 deben cumplir las siguientes propiedades [BSI22]:

    1. Las propiedades estadísticas de los números aleatorios se pueden describir mediante un modelo estocástico y sobre la base de este modelo estocástico, la entropía de los números aleatorios se puede estimar de forma fiable.

    2. El aumento medio de la entropía por bit aleatorio está por encima de un límite mínimo dado (cercano a 1).

3. Las señales de ruido digitalizadas se someten a pruebas estadísticas en línea, que son adecuadas para detectar defectos estadísticos inaceptables, en un período de tiempo razonable.

4. Un fallo total de la fuente de ruido se identifica inmediatamente. Por ello, no se deben admitir números aleatorios que hayan sido generados después de este tipo de fallo.

5. Si se identifica un fallo total de la fuente de ruido o defectos estadísticos inaceptables de los números aleatorios, se provoca una alarma, que va seguida de una respuesta apropiada, como puede ser el apagado de la fuente de ruido.

325. En la Tabla 5.1 se muestran las clases de los PTRNG que están autorizados.

Clase	Rec./Her.	Referencias	Notas
PTG.2	R	[BSI13b], [KS11]	85, 86
PTG.3	R	[BSI13b], [KS11]	85, 87

Tabla 5.1: Clases de generadores físicos de números aleatorios autorizados

326. Nota 85 [**Generadores**] El desarrollo y la evaluación de la seguridad de generadores físicos de números aleatorios requiere una amplia experiencia en este campo por lo que se recomienda el consejo de expertos en este campo en las primeras etapas.

327. Nota 86 [**Generador PTG.2**] Debido a la dificultad de evaluar la calidad de una verdadera fuente aleatoria, su uso debería limitarse a (re)semillar o generar *additional input* para un generador determinista de números aleatorios . No se permite el uso de un verdadero generador aleatorio puro, es decir, de un generador PTG.2, para su uso directo.

328. Nota 87 **[Generador PTG.3]** Es posible construir un generador de clase PTG.3 a partir de un generador PTG.2 mediante un postprocesado criptográfico de la salida del generador PTG.2 de forma adecuada (puede implementarse en software) [KS11]. En términos generales, este posprocesamiento debe implementar, además, un generador de números aleatorios determinista compatible con DRG.3 (ver §5.3) de modo que se agregue, al menos, tanta entropía al estado interno del generador de números aleatorios como requiera la aplicación criptográfica.

5.3. Generadores Deterministas de Números Aleatorios

329. Los generadores deterministas de números aleatorios, también conocidos como «generadores de números pseudoaleatorios», permiten obtener una secuencia de números pseudoaleatoria de prácticamente cualquier longitud a partir de un valor aleatorio de longitud fija, denominado «semilla». Para lograr esta secuencia, el estado interno del DRNG se inicializa con la semilla. En cada paso iterativo se obtiene un número aleatorio (generalmente, una secuencia de bits de longitud fija) a partir del estado interno del DRNG y de la salida.

330. Es claro que el estado interno de un DRNG debe protegerse de manera confiable contra lectura y manipulación.

331. Los generadores deterministas híbridos de números aleatorios, por su parte, actualizan el estado interno, de vez en cuando, con valores que son realmente aleatorios (actualizando la semilla). Con relación a esto, es posible utilizar varios esquemas de actualización de semillas, como por ejemplo actualizando la misma a intervalos regulares o a través de la petición de la aplicación.

332. En el caso de que se emplee un DRNG, se recomienda utilizar un generador que cumpla con las especificaciones DRG.3 o DRG.4 contra el potencial de ataque alto de acuerdo con AIS 20 [BSI13a, KS11]. Si se utilizan generadores de números aleatorios de clase DRG.3, es deseable refrescar el flujo de entropía en el estado del RNG, incluso si la calidad del DRNG no es lo suficientemente alta como para lograr el cumplimiento de la clase DRG.4.

333. La Tabla 5.2 presenta las diferentes clases de los DRNG autorizados.

Clase	Rec./Her.	Referencias	Notas
DRG.3	R	[BSI13a], [KS11]	88, 89
DRG.4	R	[BSI13b], [KS11]	88, 89, 90

Tabla 5.2: Clases de generadores deterministas de números aleatorios autorizados

334. Nota 88 [***Backward/Forward secrecy* DRG.3 y DRG.4**] Requisito de los DRG.3 y DRG.4 por el que es prácticamente imposible para un adversario calcular o adivinar predecesores o sucesores de una secuencia de números aleatorios conocida basándose en el conocimiento de la subsecuencia actual, con una probabilidad significativamente mayor de lo que sería posible sin conocer esta subsecuencia.

335. Nota 89 [***Enhanced backward secrecy* DRG.3 y DRG.4**] Requisito de los DRG.3 y DRG.4 por el que es prácticamente imposible para un adversario calcular o adivinar números aleatorios previamente generados basándose en el conocimiento del estado interno actual, con una probabilidad significativamente mayor de lo que sería posible sin conocer el estado interno.

336. Nota 90 [***Enhanced forward secrecy* DRG.4**] Requisito de los DRG.4 por el que es prácticamente imposible para un adversario calcular o adivinar los números aleatorios que se generan después de la siguiente actualización de la semilla basándose en el conocimiento del estado interno actual, con un probabilidad significativamente mayor de la que sería posible sin conocer el estado interno. Además, los generadores de la clase DRG.4 tienen más ventajas que los de clase DRG.3 con respecto a los ataques a la implementación.

337. Entre los generadores deterministas de números aleatorios autorizados se encuentran los que aparecen en la Tabla 5.3. En particular, para HMAC-DRBG y Hash-DRBG existen pruebas de conformidad con la clase DRG.3.

Esquema	Rec./Her.	Referencias	Notas
HMAC-DRBG	R	[SP800-90A], [ISO18031]	91, 92
Hash-DRBG	R	[SP800-90A], [ISO18031]	91, 92
CTR-DRBG	R	[SP800-90A], [ISO18031]	91, 92

Tabla 5.3: Generadores deterministas de números aleatorios autorizados

338. Nota 91 **[Semillas del DRBG]** La seguridad de un DRNG deriva de un adecuado semillado y resemillado de su estado interno, el cuál debe realizarse, siempre que sea posible, por un TRNG que pertenezca a las clases de AIS-31 autorizadas en esta guía. En los escenarios en los que esto no fuera posible, se podría aceptar que fuera (re)semillado por otro DRNG como parte de un *compliant seed tree*.

339. Nota 92 **[Resistencia hacia atrás del DRBG]** En los sistemas de intercambios de claves con claves efímeras que tienen que proporcionar PFS (*Perfect Forward Secrecy*), si se usa un DRNG sin *Enhanced backward secrecy*, un atacante que haya recuperado el estado actual de ese DRNG podrá vulnerar la PFS del sistema. Dicho de otro modo, el atacante que recupere el estado interno del DRNG, podrá calcular claves efímeras generadas anteriormente con este DRNG. Por lo tanto, solo se deben utilizar los DRNG que no permitan dicho cálculo hacia atrás.

5.4. Generadores No Físicos de Números Realmente Aleatorios

340. En muchas aplicaciones criptográficas, como en el comercio electrónico o el gobierno electrónico, no se dispone de un generador de números aleatorios físico, dado que la generación de los números que se precisan se lleva a cabo en ordenadores que no disponen de un hardware criptográfico certificado. En su lugar, se utilizan los denominados generadores no físicos de números aleatorios o NPTRNG (*Non-Physical True Random Number Generator*) [BSI22].

341. Como en los PTRNG, los NPTRNG también generan números realmente aleatorios, por lo que deben producir la entropía suficiente, pero no utilizan hardware dedicado, sino determinados recursos del sistema (como el tiempo del sistema, el contenido de la RAM, etc.) o interacciones con el usuario (como el movimiento del ratón, cadencia en la entrada del teclado, etc.). Los NPTRNG se utilizan, en general, en ordenadores que no se han desarrollado específicamente para aplicaciones criptográficas, como los ordenadores domésticos y de ofimática, los portátiles o los smartphones, entre otros.

342. Una forma típica de proceder con los NPTRNG es la siguiente: se generan largas cadenas de bits no deterministas, siendo la entropía por bit generalmente bastante baja. A continuación esta cadena de bits se mezcla con un estado interno. Sobre la base del estado interno, se calculan y se emiten posteriormente los números aleatorios.

343. En [KS11] se define una clase de funcionalidad para tales generadores de números aleatorios, denominada NTG.1. La clase de estos generadores de números aleatorios

debe estimar de manera confiable la cantidad de entropía recolectada durante su uso operativo y los datos de salida deben tener una entropía mínima.

344. La Tabla 5.4 presenta la clase de los NPTRNG autorizada.

Clase	Rec./Her.	Referencias	Notas
NTG.1	R	[KS11]	93, 94

Tabla 5.4: Clase de generador de números aleatorios ni físico ni determinista autorizado

345. Nota 93 **[Números previamente generados de NPTRNG]** Es virtualmente imposible para un adversario calcular o adivinar los números aleatorios previos basándose en el conocimiento del estado interno y las cadenas de bits aleatorias utilizadas previamente para actualizaciones de semillas, con una probabilidad significativamente mayor de lo que sería posible sin conocer el estado interno.

346. Nota 94 **[Fuentes de entropía para NPTRNG]** Para los NPTRNG es de enorme importancia que las fuentes de entropía utilizadas por el RNG no puedan ser manipuladas por un adversario en términos de reducción de la entropía o que sean predecibles si el adversario está equipado con información precisa sobre el entorno de ejecución.
Cuando se planea usar un NPTRNG como el único RNG de un sistema dado que va a ser empleado para el procesamiento de datos sensibles, siempre se debe consultar a un experto.

347. Ya se ha mencionado que para inicializar un DRNG se precisa una semilla con una entropía suficientemente alta (ver §5.3). Por ello, la semilla debe generarse con un generador físico de números aleatorios de las clases de funcionalidad PTG.2 o PTG.3. Como en los ordenadores normales no se dispone de un PTRNG o tal RNG no está certificado por una entidad independiente del fabricante, se recomienda el uso de un generador de números aleatorios ni físico ni determinista. Para este propósito, los RNG que cumplen con la clase NTG.1 son adecuados, dado que presentan un alto potencial de ataque.

Nota 95 [**Normas generales sobre el uso de RNG**] A modo de resumen, conviene considerar unas normas generales a la hora de generar números aleatorios para aplicaciones criptográficas [BSI22]:

- Cuando se usa un PTRNG se recomienda usar uno de clase PTG.3. En particular en los casos de generación de claves efímeras para esquemas de firmas digitales y en el protocolo de Diffie-Hellman. En otros contextos se puede construir un generador PTG.3 mediante el postprocesamiento criptográfico de la salida de un generador PTG.2.
- En general, los generadores PTG.3 y DRG.4 tienen, en comparación con los generadores PTG.2 y DRG.3, la ventaja de que resisten mejor a los ataques de canal lateral y ataques por inducción de fallos.
- Cuando se utiliza un generador de números aleatorios determinista, se recomienda utilizar un generador DRG.3 o DRG.4, y entonces, es recomendable generar la semilla a partir de un PTRNG de clase PTG.2 o PTG.3. Si no se dispone de un PTRNG de este tipo, se puede considerar el uso de un NPTRNG de clase NTG.1 (ver §5.4).
- Los generadores de números aleatorios híbridos combinan las propiedades de seguridad de los generadores PTRNG y DRNG. Además de una fuerte fuente de ruido, estos generadores deben estar dotados de un potente postprocesamiento criptográfico con memoria. Esto se logra típicamente postprocesando criptográficamente los números aleatorios de un generador de números aleatorios conforme a PTG.2 de una manera apropiada.

348.

5.5. Generación de Números Aleatorios con una Distribución Específica

349. Muchos mecanismos criptográficos asimétricos requieren generar números enteros que sigan una distribución específica. Para tales mecanismos, no se puede utilizar directamente un generador de bits aleatorios, dado que no ofrece directamente una distribución adecuada. Por tanto, es necesario implementar generadores de números aleatorios que ofrezcan distribuciones de salida específicas a partir de generadores de bits aleatorios.

350. La Tabla 5.5 muestra los esquemas autorizados para generar números enteros aleatorios módulo un número q dado, que no es una potencia de 2.

Algoritmo 1 Método 1 para cálculo de valores aleatorios entre $[a, b]$.

Entrada: $a, b \in \mathbb{Z}$ tal que $2^{n-1} \leq (b-a) \leq 2^n - 1$, siendo $n \in \mathbb{N}$.
Salida: $k \in [a, b]$ distribuido uniformemente.

1. Se selecciona un número $k' \in [0, 2^n - 1]$ distribuido uniformemente.
2. Si $k' > (b-a)$ repetir paso 1.
3. $k = k' + a$.
4. Devolver k.

Algoritmo 2 Método 2 para cálculo de valores aleatorios entre $[a, b]$.

Entrada: $a, b \in \mathbb{Z}$ tal que $2^{n-1} \leq (b-a) \leq 2^n - 1$, siendo $n \in \mathbb{N}$.
Salida: $k \in [a, b]$ (casi) distribuido uniformemente.

1. Se selecciona un número $k' \in [0, 2^{n+64} - 1]$ distribuido uniformemente.
2. $k = (k' \bmod (b - a + 1)) + a$.
3. Devolver k.

Esquema	Rec./Her.	Referencias	Notas
Técnica de "prueba"	R	Algoritmo 1	
Técnica "extra aleatoria"	R	Algoritmo 2	96

Tabla 5.5: Esquemas autorizados de generación de números enteros aleatorios en un intervalo $[a, b]$

351. La técnica de "prueba" asegura la generación uniforme modulo q a costa del uso de una cantidad variable de aleatoriedad, posiblemente adicional. Por su parte, la técnica "extra aleatoria" hace que los sesgos sean insignificantes a costa de una pequeña cantidad fija de aleatoriedad adicional.

352. Nota 96 **[Reducción modular aleatoria]** Debe tenerse en cuenta que el método de generación de números enteros que consiste en usar el generador de bits aleatorios subyacente para obtener un número entero aleatoriamente de manera uniforme en un rango de longitud 2^n, donde $n = \lfloor \log_2(q) \rfloor$ es el suelo (floor) o $n = \lceil \log_2(q) \rceil$ el techo (ceil) de $\log_2(q)$, posiblemente aplicando una reducción modulo q al resultado, introduce sesgos en la generación que pueden dar lugar a ataques.

6. Gestión de Claves

353. En general, la seguridad de los mecanismos criptográficos se basa en la confidencialidad, integridad y autenticidad (CIA) de las claves utilizadas. Comúnmente se acepta que para que el mecanismo empleado sea seguro, es indispensable que los adversarios no sean capaces de comprometer o alterar las claves. En esta sección se presentan los principales aspectos a tener en cuenta en cuanto a la gestión de las claves se refiere.

354. Dado que los mecanismos criptográficos autorizados se consideran robustos, su uso no pone en riesgo la CIA de las claves que utiliza. No obstante, cuando se evalúa un producto que implementa mecanismos criptográficos, se deben considerar todas las formas en las que el producto manipula el material clave y cualquier forma en la que un adversario podría intentar vulnerarlo, de modo que quede asegurado el hecho de que no puede obtener las claves.

355. Como los criptosistemas simétricos suelen estar restringidos a un grupo de cerrado de usuarios, las claves simétricas deben distribuirse entre ellos de modo que nadie externo al grupo cerrado pueda tener conocimiento de las mismas. También es fundamental que el canal de distribución de las claves esté protegido para la autenticidad e integridad.

356. En el caso de los criptosistemas asimétricos, como una de las claves es pública, puede enviarse o compartirse a través de un canal no confidencial, aunque debe protegerse para garantizar su autenticidad e integridad. Por el contrario, como la clave privada se puede generar localmente, debe estar protegida para evitar que pueda acceder a ella cualquier otro usuario que no sea su legítimo propietario.

357. La norma básica en la gestión de claves es la siguiente:

358.

> Nota 97 **[Gestión de claves]** La gestión de claves por parte del producto no debería permitir a un atacante recuperar información sobre claves secretas y privadas utilizadas para proteger la información del usuario, ni alterar o inyectar claves públicas utilizadas para proteger identidades.

6.1. Generación de Claves

359. Para que un adversario no tenga conocimiento a priori de las claves utilizadas por un determinado mecanismo criptográfico, dichas claves deben ser impredecibles. Además, se requiere que las claves sean lo suficientemente largas como para garantizar la CIA de los protocolos que las utilicen, así como que la distribución de la salida del proceso utilizado para generarlas no se pueda distinguir de una distribución uniforme.

360. En esta sección se presentan los métodos de generación de claves autorizados para los mecanismos criptográficos genéricos que requieran claves. Salvo que se indique lo contrario, las claves utilizadas para los mecanismos criptográficos convenidos de los apartados anteriores se obtendrán truncando una secuencia de bits de salida por un método de generación autorizado y dependiente del tamaño de la clave del mecanismo.

361. En la Tabla 6.1 se presentan los métodos autorizados para la generación de claves genéricas y algunas notas relacionadas.

Método	Notas
Generador de bits aleatorios autorizado (5.3)	98
Mecanismo de establecimiento de claves autorizado (3.2.4)	99, 100
Función de derivación clave autorizada (2.2.7)	100

Tabla 6.1: Métodos autorizados para la generación de claves genéricas

362. Nota 98 **[Modos de generación desde un DRNG]**

- Para claves simétricas utilizar un DRNG (5.3)
- Para claves RSA se deberían seguir las indicaciones de §A.3.
- Para claves basados en mecanismos FF-DLOG y EC-DLOG se deberían seguir las indicaciones de la sección 5.5.

363. Nota 99 **[Establecimiento de clave]** Una clave establecida a través de un procedimiento de acuerdo o establecimiento de claves no debe usarse directamente de la fuente de salida, sino que debe ser posprocesada, en primer lugar, por una función de derivación de claves autorizada (ver la sección 2.2.7).

364. Nota 100 **[Semilla de generación clave]** Algunos mecanismos de generación de claves se basan en secretos preexistentes, como por ejemplo, los exponentes efímeros en el caso del protocolo de establecimiento de claves de Diffie-Hellman, o de un secreto maestro en el caso de la generación de claves del protocolo de registro TLS. En cualquier caso, la entropía de los secretos preexistentes será de, al menos, 125 bits, o de 188 bits en caso de que se requiera resistencia a la computación cuántica.

365. Para aquellos mecanismos que tengan necesidades específicas a la hora de generar sus claves, o bien se especifica cómo generarlas en el momento en el que son tratados en esta guía, o bien se suele definir un procedimiento específico de generación de claves utilizando un generador de bits aleatorios a modo de caja negra.

6.2. Almacenamiento y Transporte de Claves

366. Nota 101 **[Almacenamiento y transporte de claves]** Cuando se transmita o almacene una clave por un medio o en un medio que no sea de confianza, la clave se protegerá con confidencialidad e integridad con un mecanismo de protección de claves autorizado, como se describe en la Sección 2.2.6.

6.3. Uso de Claves

367. El uso de una misma clave para diferentes mecanismos con el fin de garantizar, por ejemplo, la CIA, es una fuente de errores y también puede abrir vías de ataque que exploten los diversos contextos de uso de dicha clave. Hay que tener en cuenta que el uso de una clave debe entenderse en términos del objetivo de seguridad logrado y no restringido en términos de mecanismos criptográficos. Así, por ejemplo, un contexto de firma digital de un mensaje y un esquema de autenticación asimétrica pueden utilizar el mismo esquema de firma digital, pero los pares de claves utilizados en los dos contextos deben ser diferentes.

368. Nota 102 **[Uso de claves]** Una misma clave no debe utilizarse con diferentes mecanismos.

369. Nota 103 [**Plataforma de confianza**] Una clave solo se utilizará para realizar cálculos en una plataforma confiable. Se garantizará la confidencialidad e integridad de las claves secretas y privadas, y la integridad y autenticidad de las claves públicas.

370. Nota 104 [**Distribución**] La distribución de claves secretas y privadas se limitará al entorno de confianza que haga un uso efectivo de la clave.

6.4. Destrucción de Claves

371. Nota 105 [**Destrucción de claves**] Al final del ciclo de vida de una clave, la clave se borrará de forma segura de la plataforma de confianza en la que se utilizó.

372. Ya se ha mencionado que algunos esquemas criptográficos utilizan claves efímeras. En el contexto de los esquemas de establecimiento de claves de Diffie-Hellman, estas claves efímeras son intercambiadas por las partes y se utilizan para derivar un secreto compartido. Siguiendo esta derivación, las claves efímeras ya no son útiles y deben borrarse de forma segura.

373. El proceso de borrado debe adaptarse al entorno y tener en cuenta los problemas de remanencia de la memoria.

7. Autenticación de Personas

374. La autenticación de una persona (o identificación) ante otra consiste en probar de modo fehaciente que se es quien dice ser, de modo que se convenza a la otra parte de tal hecho, esto es, se demuestra la identidad de la primera persona ante la otra entidad, que puede ser una persona, un equipo, etc.

7.1. Autenticación

375. En general, la autenticación de una persona ante una entidad se asegura mediante el uso de mecanismos criptográficos. Sin embargo, si la persona debe identificarse ante sí misma en un sistema de información, entonces hay algunas diferencias.

376. La primera de ellas es que dicha persona no puede hacer uso directo de mecanismos criptográficos. La segunda es que lo más probable es que el procedimiento de autenticación se reproduzca, como es el caso, por ejemplo, de las autenticaciones a largo plazo basadas en contraseñas. En tercer lugar, suele suceder que la entropía de los datos usados para la identificación (como contraseñas), es inferior a lo que se esperaría de un sistema criptográfico estándar.

377. Por ello, los procedimientos de autenticación de personas solo se pueden realizar localmente en una plataforma confiable o a través de un canal confiable. Por ejemplo, introduciendo un número de identificación personal o PIN (Personal Identification Number) o proporcionando una *cookie* a un sitio web. Además, estos procedimientos deben requerir una interacción con el sistema porque si los datos de verificación de la identidad pudieran extraerse del sistema, un atacante podría recuperar los datos de identificación mediante el uso de fuerza bruta.

378. En esta guía nos limitaremos a considerar soluciones de autenticación de personas basadas en contraseñas y en PIN.

7.2. Procedimientos de Autenticación

379. El sistema de autenticación de personas debe imponer determinados límites en el procedimiento de autenticación, de modo que una persona no autorizada pueda llevar a cabo infinitas o un número muy elevado de pruebas hasta lograr una autenticación fraudulenta y llegue a suplantar a un usuario legítimo del sistema.

380. En esta guía consideraremos los dos casos que se detallan a continuación.

7.2.1. Limitación en el Número de Ensayos

381. En el primer caso, el sistema puede imponer un límite determinado en la cantidad de intentos que la persona que realiza la autenticación puede llevar a cabo para introducir la información correcta.

382. En general, este caso se implementa mediante recursos criptográficos, por ejemplo, tarjetas inteligentes o módulos de seguridad hardware (HSM o *Hardware Security Module*), con el fin de desbloquear el acceso a operaciones que afecten a claves almacenadas de forma segura en el recurso.

383. Después de un número determinado de intentos erróneos, la autenticación se desactiva, esto es, se vuelve imposible porque la cuenta personal ya no se puede desbloquear sin recurrir a un procedimiento administrativo. En definitiva, se trata de reducir, en la medida de lo posible, la probabilidad de aceptación falsa, es decir, impedir que una persona que no conoce los datos de identificación pueda lograr autenticarse mediante ensayos aleatorios. En general se recomienda utilizar una probabilidad máxima de falsa aceptación de 5×10^{-6}, correspondiente a un máximo de 5 intentos de un PIN de 6 dígitos. Una probabilidad máxima de aceptación falsa de 5×10^{-4}, como máximo 5 intentos de un PIN de 4 dígitos, es aceptable en aplicaciones heredadas.

384. En la Tabla 7.1 se presentan las probabilidades máxima de falsa aceptación para el caso de los intentos que se citan.

Nº de intentos	Probabilidad de falsa aceptación	Rec./Her.
5	5×10^{-6}	R
5	5×10^{-4}	H

Tabla 7.1: Probabilidades máximas de falsa aceptación

7.2.2. Limitación Temporal en el Número de Ensayos

385. En el segundo caso, se trataría de limitar el tiempo que el usuario dispondría para llevar a cabo su identificación. Esta limitación no es práctica en aplicaciones reales porque no se discrimina la velocidad con la que un usuario puede llevar a cabo diferentes intentos. Cabría la posibilidad de que el sistema limitara la velocidad a la que la persona que se intenta autenticar puede someterse a tal procedimiento. Dicho de otro modo, este caso proporciona una solución de peor calidad que el caso anterior al limitar la cantidad de intentos de autenticación por unidad de tiempo.

386. La implementación de este caso suele considerarse solo si no es práctico aplicar un mecanismo estricto de bloqueo de cuenta cuando se llega al número máximo de errores de autenticación permitidos. Un ejemplo de dicho mecanismo de limitación sería el de duplicar la demora después de cada intento de autenticación fallido antes

de que sea posible un nuevo intento. Por el momento, esta guía no considera ningún requisito para este segundo caso.

ANEXOS

A. Generación de Primos y Claves RSA

387. En este anexo se muestran diferentes métodos para la generación de números primos de determinada longitud. Tal generación es básica puesto que la generación de primos de modo seguro es utilizada en varios mecanismos de clave asimétrica, en especial en el RSA.

A.1. Generación de Números Primos

388. La generación de parámetros para los mecanismos asimétricos y de claves para el RSA necesita generar números primos, cuyo conjunto se denotará por $\mathbb{P}$, que sean aleatorios y que, en general, deben mantenerse en secreto. Por ello, esta generación debe ser aleatoria y lo más eficiente posible, de modo que la estrategia utilizada para generar tales números primos es la siguiente: en primer lugar, se genera al azar un número entero candidato del tamaño que se requiera. A continuación se prueba si tal número cumple determinadas propiedades y si es primo. En el caso de que tales pruebas fallen, el número entero candidato se actualiza y se prueba de nuevo con el siguiente candidato.

389. La función de actualización cuando se eligen diferentes números candidatos debe ofrecer un equilibrio entre la cantidad de aleatoriedad extra necesaria y la proximidad de la distribución de números primos generada a la distribución uniforme de números primos. Por otra parte, la prueba de primalidad utilizada puede ser una prueba de pseudo-primalidad, esto, es una prueba de verificación de pseudo-primos o una prueba de primos demostrables o comprobables.

390. En esta sección se muestran dos algoritmos autorizados que permiten generar números primos (véanse los Algoritmos 3 y 4). En ambos casos se supone que se dispone de una primera prueba `Test`, que verifica las condiciones que debe verificar el primo y de una segunda, `TestPrime`, que es una prueba de primalidad.

391. El primer algoritmo para generar números primos (ver Algoritmo 3) no es excesivamente eficiente y utiliza pruebas como las mencionadas anteriormente, esto es, `Test` y `TestPrime`.

392. El segundo algoritmo para generar números primos (ver Algoritmo 4) es más eficiente que el anterior. También en este caso se dispone de pruebas similares a las mencionadas anteriormente, `Test` y `TestPrime`.

393. La Tabla A.1 muestra dos métodos de generación de primos por muestreo de rechazo autorizados. El segundo de ellos es más eficiente que el primero.

Algoritmo 3 Generación de primos por muestreo de rechazo (método 1)

Entrada: un intervalo $I = [a, b] \cap \mathbb{N}$, en el que se encuentra el primo, una prueba opcional `Test` que codifica propiedades adicionales del primo a generar y una prueba de primalidad `TestPrime`.
Salida: un número primo p.

1. Se selecciona un número p de acuerdo con la distribución uniforme en I.
2. Si p no es impar, volver al paso 1.
3. Si $\texttt{Test}(p) = Falso$, volver al paso 1.
4. Si $\texttt{TestPrime}(p) = Falso$, volver al paso 1.
5. Devolver p.

Algoritmo 4 Generación de primos por muestreo de rechazo más eficiente (método 2)

Entrada: un intervalo $I = [a, b] \cap \mathbb{N}$, en el que se encuentra el primo, un entero natural pequeño B que satisfaga que el producto de los números primos menores que B, denotado por $\Pi_B = \prod_{p_i \in \mathbb{P}, p_i < B} p_i$, sea mucho menor que la amplitud del intervalo $[a, b]$, $\Pi_B \ll b - a$, una prueba opcional `Test` que codifica propiedades adicionales del primo a generar y una prueba de primalidad `TestPrime`.
Salida: un número primo p.

1. Se selecciona un número aleatorio r siguiendo la distribución uniforme en $[1, \Pi_B]$.
2. Si $\mathrm{mcd}\,(r, \Pi_B) \neq 1$, volver al paso 1.
3. Se elige al azar $k \in \mathbb{N}$ de modo que $p = k\Pi_B + r \in I$. Esto es, k se debe seleccionar de acuerdo con la distribución uniforme en $\left[\left\lceil \frac{a-r}{\Pi_B} \right\rceil, \left\lfloor \frac{b-r}{\Pi_B} \right\rfloor\right]$.
4. Si $\texttt{Test}(p) = Falso$, volver al paso 3.
5. Si $\texttt{TestPrime}(p) = Falso$, volver al paso 3.
6. Devolver p.

Esquema	Rec./Her.	Referencias	Notas
Método 1 de generación de primos	R	Algoritmo 3	106
Método 2 de generación de primos	R	Algoritmo 4	106

Tabla A.1: Generación de primos por muestreo de rechazo

394. Nota 106 (**Ataque ROCA**) El método considerado no es susceptible de ser vulnerado por el ataque ROCA (véase §A.4) [NSv+17].

A.2. Test de Primalidad y de Pseudo-Primalidad

395. Existen fundamentalmente dos procedimientos para la generación de primos: los que producen como resultado primos probables (*probable primes*) y aquellos que producen primos demostrables (*provable primes*). Aunque, en términos generales, los primeros son suficientes, en ciertos ambientes, como la generación de curvas elípticas usadas en criptografía, se pueden preferir primos demostrables [Mau95], [DGH16].

396. De manera formal, se tienen las siguientes definiciones.

397. [**Test de primalidad:**] es un algoritmo que determina que un número candidato es un número primo. Si el test de primalidad decide con total seguridad acerca de la primalidad del candidato se denomina determinista.

398. [**Test de composición:**] es un algoritmo que determina que un número candidato es un número compuesto, esto es, un test de composición solo es capaz de asegurar que el candidato es compuesto, de modo que si su salida afirma que no lo es, cabe aún la posibilidad de que lo sea. Por ello, este tipo de test sirve como test de primalidad, pero será un test probabilístico.

399. [**Número primo probable:**] es un número entero que se cree que es primo según una prueba de primalidad probabilística. No debería haber más que una probabilidad despreciable de que el llamado primo probable sea en realidad compuesto.

400. [**Número primo demostrable:**] Es un número entero que se construye para ser primo o se calcula para ser primo utilizando un algoritmo o test de primalidad determinista.

A.2.1. Generación de Primos Probables (*Probable Primes*)

401. Es bien conocido que el problema de la primalidad, esto es, decidir si un número dado es primo o no, es un problema que se puede resolver de forma determinista en tiempo polinómico [AKS04]. Si bien desde un punto de vista teórico este resultado es fundamental, el algoritmo requiere un tiempo de ejecución $\tilde{O}\left(k^{\frac{21}{2}}\right)$, siendo k el tamaño del primo generado.

402. Por este motivo en la práctica se usan métodos probabilísticos, que aunque no aseguren la primalidad del número testeado, demuestran que la probabilidad de que el número sea compuesto sea muy baja. En la Tabla A.2 se muestra el test autorizado para la determinación de la primalidad de un número candidato.

Esquema	Rec./Her.	Referencias	Notas
Test de Miller-Rabin	R	[Mil76], [Rab80]	107

Tabla A.2: Test de primalidad probabilístico autorizado

403. Nota 107 (**Primos probables**) En el caso de que la prueba de primalidad utilizada (`TestPrime`) no demuestra la primalidad del número candidato, la probabilidad de que este número sea compuesto debe ser inferior a 2^{-125}.

404. El test de Miller-Rabin (ver [Mil76], [Rab80], [MvV96, Algor. 4.24] y [DHM05, Algor. 5.28]) es el algoritmo probabilístico por excelencia. Se basa en el Teorema (pequeño) de Fermat, que afirma que para cualquier número p primo, y para cualquier número a coprimo con p, esto es, con $\text{mcd}\,(a,p)=1$, se verifica que $a^{p-1}\equiv 1 \pmod{p}$. El tiempo de ejecución esperado para el algoritmo de Miller-Rabin es $O\left((\log n)^{2+\varepsilon}\right)$, $\varepsilon \geq 1$, dependiendo del algoritmo de multiplicación empleado y siendo n el número candidato.

405. En el Teorema de Fermat, cuando tal número a existe, se denomina «base» prima con p. Si se desea comprobar la primalidad de un número candidato n y se encuentra un valor a, coprimo con n, tal que la congruencia anterior no se verifique, entonces se puede asegurar que n es compuesto. Sin embargo, la afirmación recíproca no es siempre cierta. Dicho de otro modo, el hecho de que no se encuentre base alguna en la que no se verifique dicho teorema no garantiza que el número n sea primo.

406. Es importante señalar que, aunque la condición que proporciona el Teorema pequeño de Fermat sea sólo una condición necesaria, el número de excepciones es muy pequeño. Tales excepciones se conocen como «pseudoprimos». De hecho, los números n que satisfacen la congruencia del Teorema pequeño de Fermat,

$a^{n-1} \equiv 1 \pmod{n}$, para toda base $a \in [2, n-1]$, prima con n, reciben el nombre de números de Carmichael (el más pequeño de tales números es $n = 3 \cdot 11 \cdot 17 = 561$).

407. El algoritmo de pseudo-primalidad de Miller-Rabin (véase el Algoritmo 5) utiliza como entrada los números $n, t \in \mathbb{N}$, con $n \geq 3$ e impar y t un parámetro de seguridad, y determina si n es primo con una probabilidad de acierto de $1 - \frac{1}{2^{2t}} = 1 - 2^{-2t}$.

Algoritmo 5 Algoritmo de pseudo-primalidad de Miller-Rabin

Entrada: número entero candidato, n, y parámetro de seguridad (número de iteraciones), t.
Salida: decisión sobre la primalidad del candidato n.

1. Se determinan $s, r \in \mathbb{Z}$ de modo que $n - 1 = 2^s r$, con r impar.

2. **for** i **from** 1 **to** t **do**
   Se elige al azar un entero a, con $2 \leq a \leq n - 2$
   $b \leftarrow a^n \pmod{n}$
   if $(b \neq 1$ and $b \neq n - 1)$ **then do**
   $j \leftarrow 1$
   while $j \leq s - 1$ and $b \neq n - 1$ **do**
   $b \leftarrow b^2 \pmod{n}$
   if $b = 1$ **then** devuelve "Compuesto"
   $j \leftarrow j + 1$
   if $b \neq n - 1$ **then** devuelve "Compuesto"

3. Devuelve "Primo con probabilidad $(1 - 2^{-2t})$".

408. Si P_{obj} es el valor objetivo de la probabilidad considerada, entonces se tiene que $P_{obj} \geq 2^{-2t}$, es decir, tomando logaritmos y despejando

$$\log_2 (P_{obj}) \geq -2t \log_2 2,$$
$$t \geq -\frac{1}{2} \log_2 (P_{obj}),$$

es decir, $1 \leq t \leq \lfloor -\log_2 (P_{obj}) \rfloor$.

409. Se puede estimar la probabilidad, $P_{k,t}$, de que un número entero impar de k bits que pase t iteraciones del test de Miller-Rabin sea en realidad compuesto. Esta probabilidad se entiende como la relación entre el número de números compuestos impares de k bits que se puede esperar que pasen t rondas de pruebas de este test, con bases generadas aleatoriamente, con la suma de ese valor y el número de primos impares enteros de longitud binaria k [DLP93], [FIPS186-5, App. C.1].

410. Esta probabilidad viene dada por la siguiente expresión:

$$P_{k,t} = 2{,}00743 \cdot \ln(2) \cdot k \cdot 2^{-k} \cdot \left(2^{k-2-M\cdot t} + \frac{8}{3}(\pi^2 - 6)2^{k-2} \sum_{m=3}^{M} 2^{m-(m-1)t} \sum_{j=2}^{m} \frac{1}{2^{j+\frac{k-1}{j}}}\right), \quad \text{(A.1)}$$

donde $3 \leq M \leq \left\lceil 2\sqrt{k-1} - 1 \right\rceil$. En definitiva, se acepta el valor del número de iteraciones, t, cuando se cumpla la condición $P_{k,t} \leq P_{obj}$.

411. A modo de ejemplo, en la Tabla A.3 se presenta el número de iteraciones, t, a realizar con el test de Miller-Rabin[10] en función de la longitud en bits, k, de la entrada impar generada aleatoriamente para lograr probabilidades menores que 2^{-125}.

	$P_{obj} = 2^{-125}$
Nº iter., t	Long. bits candidato, k
6	$950 \leq k < 1036$ (1041)
5	1036 (1041) $\leq k <$ 1289 (1297)
4	1289 (1297) $\leq k <$ 1720 (1729)
3	1720 (1729) $\leq k <$ 2607 (2626)
2	2607 (2626) $\leq k <$ 5672 (5701)
1	5672 (5701) $\leq k$

Tabla A.3: Número de iteraciones del test de Miller-Rabin según la longitud en bits del candidato para $P_{obj} = 2^{-125}$

412. Según la Tabla A.3, obtenida haciendo uso de la fórmula (A.1), se deduce que serán necesarias $t = 6$ rondas con $k = 1024$, o $t = 3$ rondas con $k = 2048$ para para excluir, con una probabilidad de error de 2^{-125}, que p es un número compuesto, aunque el algoritmo de Miller-Rabin identifique a p como un número primo.

413. Dado que el mínimo número de bits exigido para un primo en el caso del criptosistema RSA es de 1536 (el módulo tendrá un tamaño de 3072 bits) y que para el algoritmo de Diffie-Hellman el tamaño del primo es de 3072 bits, siguiendo la expresión (A.1), para $k = 1536$, se necesitan $t = 4$ iteraciones y si $k = 3072$, el número de iteraciones es $t = 2$.

[10] El número de iteraciones y las longitudes en bits de los candidatos mostrados en la Tabla A.3 las hemos computado a la hora de elaborar dicha Tabla. En el caso de $P_{obj} = 2^{-125}$, se muestran entre paréntesis los valores publicados en [ACM1.3]. Suponemos que la diferencia entre los valores calculados y los publicados para este caso, se debe a la distinta precisión numérica empleada a la hora de calcular los valores correspondientes aplicando la fórmula (A.1).

A.3. Generación del Par de Claves de RSA

414. A continuación se presenta un algoritmo (véase el Algoritmo 6) que permite generar los dos números primos que se emplean para determinar el módulo en el mecanismo asimétrico RSA (véase la sección 3.1.1). El par de números primos aleatorios deben satisfacer un conjunto de restricciones para asegurar el tamaño de la clave, la consistencia del par de claves pública/privada y evitar claves débiles.

Algoritmo 6 Generación del par de claves para el mecanismo asimétrico RSA

Entrada: longitud en bits del módulo RSA, k, y exponente público (cifrado), e.
Salida: dos números primos p, q.

1. Si e es par o si $e \leq 2^{16}$, muestra *Fallo*.
2. Usando un método de generación de primos aleatorios autorizado (ver Algoritmos 3 y 4), se genera un primo aleatorio p en el intervalo $\left[\frac{1}{\sqrt{2}}2^{\frac{k}{2}}, 2^{\frac{k}{2}}\right]$, de modo que $\texttt{Test}(p)$: $\text{mcd}\,(p-1, e) = 1$.
3. Usando un método de generación de primos aleatorios autorizado (ver Algoritmos 3 y 4), se genera un primo aleatorio q en el intervalo $\left[\frac{1}{\sqrt{2}}2^{\frac{k}{2}}, 2^{\frac{k}{2}}\right]$, de modo que $\texttt{Test}(q)$: $\text{mcd}\,(q-1, e) = 1$ y $|p-q| \geq 2^{\frac{k}{2}-100}$.
4. Se determina $d = e^{-1} \pmod{\text{mcm}\,(p-1, q-1)}$.
5. Si $d \leq 2^{\frac{n}{2}}$, volver al paso 1.
6. Devolver p, q.

415. En la Tabla A.4 se presenta el algoritmo autorizado para la generación del par de claves para el mecanismo asimétrico RSA.

Esquema	Rec./Her.	Referencias	Notas
Método generación de claves RSA	R	Algoritmo 6	108, 109, 110

Tabla A.4: Algoritmo autorizado para la generación claves RSA

416. Nota 108 (**Generación de claves RSA**) Los números primos p y q deben ser dos primos generados aleatoriamente de la misma longitud y cuyo producto (módulo RSA) debe tener la longitud de bits dada. Los dos números primos no deben ser demasiado cercanos para evitar ataques de factorización que exploten una posible distancia pequeña entre los dos factores, por lo que debería verificarse que $|p-q| \geq 2^{\frac{k}{2}-100}$, siendo k el tamaño de dichos números primos.

417. Nota 109 (**Tamaño de la clave privada, d**) El tamaño de la clave privada, d, debe ser lo suficientemente grande, es decir, se debe verificar que $d > 2^{\frac{k}{2}}$, donde k la longitud de bits del módulo RSA. Debe tenerse en cuenta que esta condición está garantizado si el exponente público, e, es pequeño [HMQ03].

418. Nota 110 (**Intervalo de los números primos**) Para garantizar la seguridad de los pares de claves pública/privada para los que el módulo RSA se ha obtenido multiplicando dos números primos generados de forma independiente con uno de los métodos autorizados (ver la Tabla A.1), es importante que el intervalo considerado, $I = [a, b] \cap \mathbb{N}$ no sea demasiado pequeña. De hecho, si se considera que el módulo tiene una longitud de k bits, se suele considerar $I = \left[\left\lfloor \frac{2^{\frac{n}{2}}}{\sqrt{2}} \right\rfloor, \left\lceil 2^{\frac{n}{2}} \right\rceil\right] \cap \mathbb{N}$. Las diferentes elecciones del intervalo I cumplen con esta guía si p y q se extraen del mismo intervalo I y si $\#(I) \geq 2^{-8}b$.

A.4. Ataque ROCA al Algoritmo RSA

419. Desde 2016 han aparecido algunas publicaciones que plantean la cuestión de si los bits de una clave pública RSA, (n, e), pueden revelar información sobre la elección realizada en el diseño e implementación del algoritmo que genera los números primos p y q, es decir, si es posible identificar el origen de los primos conociendo sólo el módulo RSA y no su factorización.

420. De hecho, en [Nem16], [vNS+16b] y en [vNS+16a], los autores intentan verificar si los pares de claves RSA generados por cierto software y determinadas tarjetas inteligentes ofrecen la calidad y la seguridad requerida en relación con la aleatoriedad deseada y la resiliencia frente a los ataques más generalizados. Para ello, estudian si es posible determinar el origen de las claves, es decir, identificar qué librería de software o qué tarjeta inteligente es la responsable de generar los números primos asociados a una clave RSA, suponiendo que solo se conoce la clave pública. Así, Švenda et al. [vNS+16b] demostraron que las opciones de implementación en las bibliotecas criptográficas permiten suponer el origen de las claves RSA públicas.

421. Posteriormente, Nemec et al. informaron en [NSv+17] sobre su descubrimiento de un fallo en el algoritmo en la construcción de números primos para la generación de claves RSA en una biblioteca ampliamente utilizada de un importante fabricante de hardware criptográfico. Este ataque ha sido llamado «ataque ROCA» debido al título del artículo. Los números primos generados por la biblioteca sufren una importante pérdida de entropía, defecto que aprovecharon los autores para proponer

un método práctico de factorización para varias longitudes de clave, incluidos 1024 y 2048 bits. Su método no requería información adicional excepto conocer el valor del módulo RSA y no dependía de un generador de números aleatorios débil o defectuoso. Nemec et al. extendieron el ataque de factorización de Coppersmith utilizando una forma alternativa de los números primos en cuestión. La librería software utilizada se encontraba en dispositivos con certificación CC EAL 5+, que son utilizados en una extensa gama de aplicaciones reales, incluidas tarjetas de identidad, pasaportes y tokens para autenticación o firma de software.

422. Se identificaron, de esta forma, decenas de miles de claves con esta debilidad. Los autores estimaron que la cantidad de dispositivos afectados era del orden de decenas de millones. Además, en el peor de los casos, la factorización de claves de 1024 y 2048 bits precisaba de menos de 3 meses de CPU y 100 años de CPU en un solo núcleo de CPU, respectivamente. Pero además, todas las claves susceptibles de ser atacadas contienen una característica digital que es verificable en microsegundos en un ordenador común, por lo que todas las claves vulnerables se pueden identificar de forma inmediata.

423. Los resultados del ataque ROCA obligó a varios gobiernos europeos a revocar todos los certificados digitales de millones de tarjetas de identificación de sus ciudadanos, ya que tenían claves de 1024 bits y se podía suplantar la identidad de sus ciudadanos. En España se optó por la misma medida de prevención, aunque los DNIe españoles no corrían el mismo peligro que los documentos de identidad utilizados en otros países, ya que el DNIe español utiliza claves de 2048 bits.

B. Criptografía Postcuántica

424. En este anexo se aborda la criptografía conocida como postcuántica. A pesar de que en el momento de escribir y publicar esta guía no ha concluido el proceso para la determinación de lo que serán, previsiblemente, los nuevos estándares del NIST de la criptografía, resistentes a la amenaza que supone la computación cuántica, parece oportuno señalar cuáles son sus premisas y puntos de partida y cuáles son las propuestas candidatas que optan con mayores posibilidades a constituirse como nuevos estándares. En la fecha de la publicación de esta guía, el NIST ha publicado la lista de los algoritmos seleccionados que han superado la tercera ronda y ha abierto una cuarta ronda. De los algoritmos seleccionados, uno pertenece a la categoría PKE/KEM, se trata de CRYSTALS-Kyber, y los restantes son de la categoría de firmas digitales, son CRYSTALS-Dilithium, Falcon y SPHINCS$^+$. Con la apertura de una cuarta ronda, el NIST no ha descartado completamente otras propuestas, de modo que otros cuatro algoritmos deberán seguir siendo analizados. Se trata de BIKE (*Bit Flipping Key Encapsulation*), HQC (*Hamming Quasi-Cyclic*), Classic McEliece y SIKE (*Supersingular Isogeny Key Encapsulation*). Se hará especial hincapié en los algoritmos seleccionados que han superado la tercera ronda y se tendrán en cuenta los que aún deberán ser analizados en la cuarta. Los primeros ya han sido incluidos en los capítulos anteriores que les corresponden, como mecanismos autorizados.

B.1. La Amenaza Cuántica

425. En esta sección se hará una breve introducción a las razones que han dado lugar al nacimiento de la denominada criptografía postcuántica (*Post-Quantum Cryptography*, PQC) , cuyo principal objetivo es el de resistir a la amenaza real que supone la enorme potencia de los ordenadores cuánticos. Por otra parte, a lo largo de las siguiente secciones se describirán, brevemente, las herramientas matemáticas y los correspondientes problemas matemáticos[11] utilizados para garantizar la seguridad de estos nuevos protocolos.

426. A día de hoy, la comunidad criptográfica ha aceptado que la computación cuántica constituye una seria amenaza para la criptografía actual [CLM20]. De hecho, es bien sabido que los algoritmos cuánticos propuestos por Shor [Sho97] permiten vulnerar los principales mecanismos asimétricos utilizados hoy en día. La cuestión es que si se desarrolla un ordenador cuántico con suficiente capacidad de cómputo, los problemas matemáticos en los que se basa la seguridad de estos mecanismos, como la factorización de números enteros (véase §3.1.1) y el cálculo de logaritmos

[11] Consideramos que la escasa divulgación y la falta de conocimiento por no especialistas de estas herramientas y problemas —en algunos casos, más complejos que los empleados hasta ahora— hace necesaria la inclusión de unas definiciones preliminares que no obliguen al lector a tener que recurrir a libros especializados.

discretos (véanse §3.1.2 y §3.1.3) podrían resolverse solo en unas pocas horas. De hecho, si un ordenador actual necesita $\mathcal{O}\left(2^{\sqrt[3]{\log n}}\right)$ operaciones bit para romper un algoritmo, un ordenador cuántico, usando el algoritmo de Shor, reduciría ese número de operaciones bit a $\mathcal{O}\left(\log^3 n\right)$ con un almacenamiento de memoria de $\mathcal{O}(\log n)$ bits.

427. En relación con los mecanismos simétricos (véase §2.1), los algoritmos de Grover [Gro96], [Gro97] y Simon [Sim97] reducirían el tiempo de cálculo necesario para romperlos a la raíz cuadrada del tiempo actual. Esto es, si se desarrolla un ordenador cuántico con la capacidad de cómputo suficiente, la seguridad de los mecanismos simétricos actuales sería equivalente a la de los mismos mecanismos con claves de longitud la mitad. Dicho de otro modo, si un PC actual necesita $\mathcal{O}(n)$ operaciones bits para romper uno de estos mecanismos, con el algoritmo de Grover este tiempo se reduciría a $\mathcal{O}\left(\sqrt{n}\right)$ operaciones bits y requeriría un almacenamiento en memoria de $\mathcal{O}(\log n)$ bits.

B.1.1. Convocatoria del NIST

428. Debido a la amenaza de la computación cuántica ya comentada, el NIST lanzó en 2017 una Convocatoria Internacional para seleccionar nuevos algoritmos criptográficos resistentes a esta computación cuántica [NIS17]. Los únicos algoritmos afectados por esta convocatoria son los cifrados asimétricos, los mecanismos de encapsulamiento de clave y las firmas digitales. En definitiva, el objetivo final de este proceso de estandarización para lograr algoritmos resistentes a la computación cuántica es el de actualizar las siguientes publicaciones del NIST: [FIPS186-5], [SP800-56A] y [SP800-56B]. Por ello en este capítulo de la guía no se tratarán los mecanismos simétricos que puedan proponerse en un futuro y que sean resistentes a la computación cuántica.

429. La seguridad de los algoritmos considerados en la convocatoria del NIST se fundamenta en determinados problemas matemáticos basados en cinco primitivas:

- Códigos correctores de errores, que dan lugar a la criptografía basada en códigos (*Coded-based cryptography*)
- Funciones resumen, que proporcionan la criptografía basada en resúmenes (*Hash-based cryptography*)
- Polinomios multivariantes cuadráticos, que dan lugar a la llamada criptografía multivariante cuadrática (*Multivariate Quadratic Cryptography*, MQC)
- Retículos, dando pie a la criptografía basada en retículos (*Lattice-based cryptography*)
- Isogenias definidas sobre curvas elípticas, que proporcionan la criptografía basada en isogenias (*Isogeny-based cryptography*).

430. En julio de 2022, el NIST publicó [NIS22] la lista de algoritmos seleccionados (a expensas de una cuarta ronda) y que listamos en las Tablas B.1 y B.2, junto con sus primitivas matemáticas asociadas.

Criptosistema asimétrico y KEM	Primitiva matemática
CRYSTALS-Kyber	Retículo, MLWE

Tabla B.1: Propuesta KEM seleccionada después de la tercera ronda y Primitiva matemática asociada

Firma digital	Primitiva matemática
CRYSTALS-Dilithium	Retículo, MLWE
Falcon	Retículo, SIS
SPHINCS$^+$	Hash

Tabla B.2: Propuestas a firmas seleccionadas después de la tercera ronda y Primitivas matemáticas asociadas

431. Como ya se han mencionado, existen cuatro algoritmos que el NIST ha mantenido para ser estudiados en una cuarta ronda. Los mismos se listan en la Tabla B.3.

Criptosistema asimétrico y KEM	Primitiva matemática
Classic McEliece	Código Goppa
BIKE	Código de densidad moderada cuasi-cíclico
HQC	Código cuasi-cíclico de Hamming
SIKE	Isogenias sobre curvas elípticas

Tabla B.3: Candidatos a KEM para ser analizados en la cuarta ronda y Primitivas matemáticas asociadas

432. En los capítulos anteriores de esta guía hemos incluido los mecanismos criptográficos que a nivel nacional se consideran autorizados, porque ofrecen una seguridad adecuada. Debe notarse que algunos de ellos podrán ser vulnerados si la computación cuántica acaba teniendo la potencia de cálculo necesaria. Por ello, su papel pasará a ser el de los nuevos estándares que se consideren tanto por el NIST como por la comunidad internacional y española.

433. Dado que el NIST está a punto de concluir y publicar los resultados de su convocatoria sobre PQC, en este capítulo solo hemos incluido las propuestas que

se han superado la tercera ronda o que siguen considerándose para ser evaluadas en la cuarta. Por ello, hemos añadido en este capítulo las primitivas matemáticas consideradas en la convocatoria del NIST. No obstante, no se ha incluido la primitiva de isogenias sobre curvas elípticas dado que, aunque en la fecha de la publicación de los candidatos que habían superado la tercera ronda, SIKE era un candidato incluido, todo apunta a que no será tenido en cuenta en el futuro, dado que se ha encontrado un ataque de recuperación de clave eficiente para SIKEp434 (nivel de seguridad 1) utilizando un procesador de un solo núcleo en, aproximadamente, una hora [CD22]. No obstante, a la anterior consideración se hará una excepción. Se trata de la propuesta FrodoKEM.

434. Para el CCN es de especial importancia el estudio de los algoritmos de acuerdo de clave, por lo que considera, además, del CRYSTALS-Kyber, seleccionado por el NIST y mostrado en la Tabla B.1, el algoritmo basado en retículos no estructurados FrodoKEM (presentado en la Tabla B.4), que puede considerarse como una de las opciones más conservadoras con relación a su seguridad.

Criptosistema asimétrico y KEM	Primitiva matemática
FrodoKEM (Round 3)	Retículo, LWE

Tabla B.4: Propuesta KEM de interés para el CCN

B.1.2. Seguridad

435. Una de las primeras consideraciones a tener en cuenta para determinar nuevos estándares para la PQC es su seguridad. Esto es, cualquier nuevo mecanismo criptográfico que se proponga debe ser seguro frente a ataques clásicos y a ataques cuánticos.

436. Las «pruebas de reducción» han sido las principales herramientas utilizadas para lograr confianza en un mecanismo criptográfico. Hablando informalmente, estas pruebas de reducción consisten en considerar un problema matemático difícil de resolver (supuestamente), de modo que la afirmación de que un mecanismo criptográfico es demostrablemente seguro frente a una definición de seguridad significa que se puede probar que romper el mecanismo criptográfico implica resolver (en tiempo polinómico) el problema difícil considerado. Dicho de otro modo, la seguridad del mecanismo queda «reducida» a la seguridad del problema matemático.

437. La seguridad clásica de un mecanismo criptográfico, para un determinado conjunto de parámetros y tamaños de clave, se estima en bits. Para los mecanismos asimétricos (criptosistemas, protocolos de acuerdo de clave y firmas digitales) se han introducido diferentes objetivos de seguridad, entre los que destacamos, de forma muy resumida, los siguientes:

- Ataque unidireccional (*One-Way Attack*, OWA): dado un texto cifrado correspondiente a un texto claro aleatorio y su clave pública, un atacante no es capaz de encontrar el texto claro original. Una versión de este ataque es el ataque unidireccional al texto claro elegido (*One-way against Chosen Plaintext Attack*, OW-CPA).

- Indistinguibilidad por ataque al texto claro elegido (*INDistinguishability under Chosen Plaintext Attack*, IND-CPA): dado el cifrado de uno de dos textos claros elegidos por un atacante y su clave pública, el atacante no es capaz de determinar cuál de los textos claros corresponde al texto cifrado dado. Esta definición se aplica a los mecanismos cuando se utilizan claves públicas de un solo uso.

- Indistinguibilidad por ataque (adaptativo) al texto cifrado elegido (*INDistinguishability under (Adaptive) Chosen Ciphertext Attack*, (IND-CCA2) IND-CCA1): el atacante bajo las condiciones del IND-CPA no es capaz de llevar a cabo el ataque con éxito incluso si además se le da acceso a un oráculo de descifrado. El oráculo se puede utilizar en todos los textos cifrados excepto en el texto cifrado del desafío.

  En la definición no adaptativa (IND-CCA1), el adversario puede consultar el oráculo solo hasta que recibe el texto cifrado de desafío. En el caso adaptativo (IND-CCA2), el adversario puede continuar consultando el oráculo de descifrado incluso después de haber recibido un texto cifrado de desafío, con la advertencia de que no puede pasar el texto cifrado de desafío para el descifrado. Estas definiciones se aplican a los mecanismos cuando se utilizan claves públicas estáticas.

- Existencialmente infalsificable bajo un ataque adaptativo al mensaje elegido (*Existentially UnForgeable under adaptive Chosen Message Attack*, EUF-CMA): un atacante no puede falsificar nuevas firmas para una clave pública dada, incluso si tiene la capacidad de ver firmas en los mensajes elegidos por el atacante.

438. La seguridad cuántica, por otra parte, mide la seguridad de un mecanismo criptográfico frente a ataques cuánticos. A día de hoy, no está claro cómo estimar la seguridad de un mecanismo criptográfico frente a ataques cuánticos.

439. Lo que se pretende con la criptografía postcuántica es que se utilice allá donde se usa la criptografía asimétrica en la actualidad, esto es, los futuros mecanismos estándares postcuánticos deberían reemplazar directamente a los actuales.

440. Para un mecanismo postcuántico (PQ) es importante considerar el tamaño de la clave, el del texto cifrado o el de la firma. En general, estos tamaños para la PQC son mucho mayores que los empleados en la actualidad, lo que puede llegar a ser un problema en aplicaciones donde la transmisión se realiza a través de canales de

comunicación con ancho de banda limitado. Por otra parte, además de la velocidad de procesamiento, también se debe considerar la velocidad en la generación de claves, en el cifrado y el descifrado, y en la generación de firma y su verificación. Para estos mecanismos también se considera como una propiedad deseada el secreto perfecto persistente y la garantía de seguridad para resistir ataques de canal lateral.

B.2. Criptografía Basada en Retículos

441. Presentaremos, en primer lugar, unas nociones elementales sobre retículos (*lattices*) y los principales problemas definidos en esta estructura, para posteriormente comentar las propuestas que han superado la tercera ronda del NIST.

B.2.1. Retículos

442. Dada una base de vectores, $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_m\}$, un «retículo», $\mathcal{L}$, es un subespacio vectorial formado por las combinaciones lineales enteras (no reales) de los elementos de la base, $\mathbf{b}_i$:

$$\mathcal{L} = \left\{ \sum_{i=1}^{m} a_i \cdot \mathbf{b}_i \colon a_i \in \mathbb{Z} \right\} = \{\mathbf{B} \cdot \mathbf{a} \colon \mathbf{a} \in \mathbb{Z}\}.$$

443. El conjunto $\mathbf{B} = \{\mathbf{b}_1, \ldots \mathbf{b}_n\}$ se denomina base del retículo. El retículo generado por la base $\mathbf{B}$ se denota por $\mathcal{L}(\mathbf{B})$.

444. Como un retículo es la versión discreta de un subespacio vectorial, se puede hablar del elemento no nulo más pequeño del retículo. Se llama mínimo no nulo (distancia mínima) de un retículo a

$$\lambda_1(\mathcal{L}) = \text{mín}\left\{||\mathbf{x}|| \colon \mathbf{x} \in \mathcal{L}, \mathbf{x} \neq 0\right\}.$$

445. Los principales problemas, computacionalmente difíciles, que se consideran en retículos se definen en las Notas 111–119.

446. Nota 111 **[Problema del vector más corto]** (*Shortest Vector Problem*, SVP) Consiste en encontrar un vector no nulo, $\mathbf{x} \in \mathcal{L}$, que sea el más corto de $\mathcal{L}$, es decir, $||\mathbf{x}|| \leq ||\mathbf{y}||$, para todo $\mathbf{y} \in \mathcal{L}$, esto es, $||\mathbf{x}|| = \lambda_1(\mathcal{L})$.

447. Nota 112 [**Problema del vector más cercano**] (*Closest Vector Problem*, CVP) Dado $\mathcal{L}$ en $\mathbb{R}^n$ y un vector $\mathbf{x} \in \mathbb{R}^n$ de modo que $\mathbf{x} \notin \mathcal{L}$, este problema consiste en encontrar un vector $\mathbf{y} \in \mathcal{L}$ de modo que $||\mathbf{x} - \mathbf{y}|| \leq ||\mathbf{x} - \mathbf{z}||$ para todo $\mathbf{z} \in \mathcal{L}$.

448. Nota 113 [**Problema de la solución entera más corta**] (*Short Integer Solution*, SIS) Dados m vectores uniformemente aleatorios $\mathbf{a}_i \in \mathbb{Z}_q^n$ que considerados como vectores columnas definen una matriz $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, este problema consiste en encontrar un vector no nulo $\mathbf{z} \in \mathbb{Z}^m$ de norma $||\mathbf{z}|| \leq \varepsilon$ tal que

$$\mathbf{A} \cdot \mathbf{z} = \sum_{i=1}^{m} \mathbf{a}_i \cdot \mathbf{z}_i = 0 \in \mathbb{Z}_q^n.$$

Dicho de otro modo, dados muchos elementos uniformemente aleatorios de un cierto grupo aditivo finito grande, el $\text{SIS}_{n,q,\varepsilon,m}$ trata de encontrar una combinación lineal de ellos, con enteros suficientemente cortos, que sume cero.

449. Nota 114 [**Problema del aprendizaje con errores**] (*Learning With Errors*, LWE) Este problema se parametriza por un entero n, un número primo $q \geq 2$ y una distribución de probabilidad χ sobre $\mathbb{Z}_q$. Típicamente χ es una distribución normal de media ν y desviación estándar δ:

$$\chi = G(x) = \frac{1}{\delta\sqrt{2\pi}} \exp^{\frac{1}{2}\left(\frac{x-\nu}{\delta}\right)^2}.$$

Una distribución $A_{\mathbf{s},\chi}$ de un problema LWE sobre $\mathbb{Z}_q^n \times \mathbb{Z}_q$ se muestrea eligiendo uniforme y aleatoriamente $\mathbf{a} \xleftarrow{\$} \mathbb{Z}_q^n$, $e \xleftarrow{\$} \mathbb{Z}_q$, y considerando como salida el par $(\mathbf{a}, b)$, siendo $b = \langle \mathbf{s}, \mathbf{a} \rangle + e \pmod{q}$.

450. Existen dos versiones del problema LWE: búsqueda y decisión, que se definen a continuación.

Nota 115 **[Problema del aprendizaje con errores (búsqueda)]** Dadas m muestras independientes $(\mathbf{a}_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ de $A_{\mathbf{s},\chi}$ con $b_i = \langle \mathbf{s}, \mathbf{a}_i \rangle + e_i \pmod q \in \mathbb{Z}_q$, se trata de encontrar el vector secreto $\mathbf{s} \in \mathbb{Z}_q^n$, siendo $m > n$.
La idea es determinar el valor $\mathbf{s}$ a partir de m muestras dadas:

451.
$$\begin{aligned} \mathbf{a}_1 &\xleftarrow{\$} \mathbb{Z}_q^n, \quad b_1 = \langle \mathbf{s}, \mathbf{a}_1 \rangle + e_1 \pmod q, \\ \mathbf{a}_2 &\xleftarrow{\$} \mathbb{Z}_q^n, \quad b_2 = \langle \mathbf{s}, \mathbf{a}_2 \rangle + e_2 \pmod q, \\ &\vdots \\ \mathbf{a}_m &\xleftarrow{\$} \mathbb{Z}_q^n, \quad b_m = \langle \mathbf{s}, \mathbf{a}_m \rangle + e_m \pmod q. \end{aligned}$$

Nota 116 **[Problema del aprendizaje con errores (decisión)]** En este problema, el objetivo es distinguir entre dos pares de vectores $(\mathbf{a}_i, b_i)$ y $(\bar{\mathbf{a}}_i, \bar{b}_i)$, donde $\mathbf{a}_i, \bar{\mathbf{a}}_i \xleftarrow{\$} \mathbb{Z}_q^n$, $b_i = \langle \mathbf{s}, \mathbf{a}_i \rangle + e_i \pmod q$ y $\bar{b}_i \in \mathbb{Z}_q$.

452. Es decir, se trata de decidir, para un par de vectores dados, si el segundo vector es el producto escalar del primer vector por algún vector secreto, $\mathbf{s}$, sumado con algún error, o si es el segundo vector es uniformemente aleatorio.

453. Es posible modificar el problema LWE dotándolo de una estructura de anillo o de módulo [Reg04], definiendo entonces nuevos problemas.

Nota 117 **[Problema LWE sobre anillos]** (Ring Learning With Errors, RLWE) Se considera el anillo de polinomios, $R = \mathbb{Z}[x]/(\ell(x))$, de grado n sobre $\mathbb{Z}$. En general se considera $\ell(x) = x^n - 1$ si n es primo o $\ell(x) = x^n + 1$ si n es una potencia de 2. Se considera además, el anillo cociente $R_q = R/qR$, siendo q impar y suficientemente grande (hacer módulo q).

454. El problema RLWE se parametriza por el anillo R, de grado n sobre $\mathbb{Z}$, un módulo entero positivo, q, que define el anillo cociente R_q, y una distribución de error, χ, sobre R.
Dado un secreto $s \in R_q$, una distribución RLWE, $A_{s,\chi}$, sobre $R_q \times R_q$ se muestrea eligiendo uniforme y aleatoriamente $a \in R_q$, con $e \leftarrow \chi$ y considerando como salida $(a, b = \langle s, a \rangle + e \pmod q)$.

Nota 118 [**Problema LWE sobre anillos (decisión)**] En este caso, se trata de distinguir entre muestras de RLWE y muestras uniformemente aleatorias. El problema se parametriza, además, con el número de muestras disponibles, m.

455. Este problema consiste en: dadas m muestras independientes $(a_i, b_i) \in R_q \times R_q$, distinguir si cada muestra se distribuye de acuerdo a una de las dos opciones siguientes: (1) $A_{s,\chi}$ para un $s \in R_q$ uniformemente aleatorio (fijo para todas las muestras), o (2) una distribución uniforme, sin una ventaja apreciable.

Nota 119 [**Problema LWE sobre módulos**] (Module Learning With Errors, MLWE) Este problema es análogo al problema RLWE pero en lugar de considerar la estructura subyacente de anillo, se considera la estructura de módulo[12].

456.

457. El problema RLWE es más compacto que el LWE debido a la estructura de anillo subyacente lo que permite implementaciones más eficientes pero, por el contrario, esta misma estructura lo hace más vulnerable a determinados ataques. Análogamente sucede con el problema MLWE que, al definirse sobre una estructura de módulo es más compacto que el LWE pero menos que el RLWE. Por ello, se considera menos vulnerable que el RLWE pero más propenso a ataques que el LWE.

B.2.2. ML-KEM

458. En agosto de 2024 el NIST publicó el estándar [FIPS203] *Module-Lattice-Based Key-Encapsulation Mechanism Standard* (ML-KEM). En esta sección presentaremos las características de este nuevo estándar como mecanismo de encapsulado de claves.

459. El esquema general de ML-KEM se muestra en la Tabla B.5. Dicho esquema cuenta con los tres algoritmos estándar de un KEM genérico: algoritmo de generación de claves, ML-KEM.KeyGen, encapsulado, ML-KEM.Encaps, y desencapsulado, ML-KEM.Decaps.

[12] La estructura de módulo sobre un anillo puede considerarse como la generalización de la noción de espacio vectorial sobre un cuerpo. En este caso, los escalares son los elementos de un anillo con elemento neutro y donde está definida una multiplicación entre elementos del anillo y elementos del módulo.

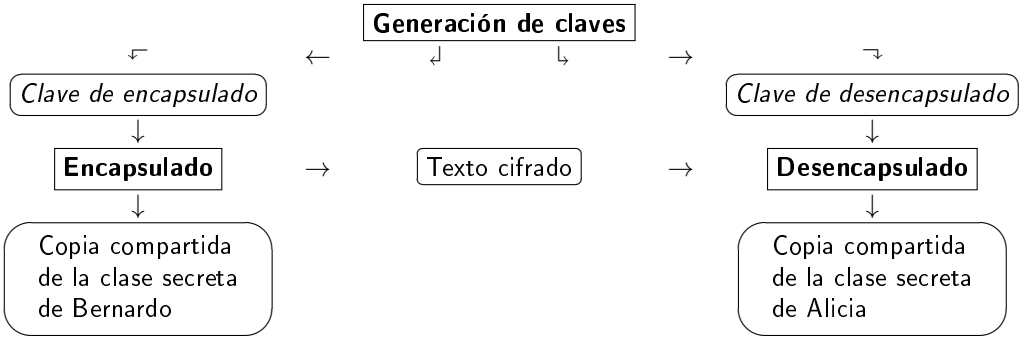


Tabla B.5: Esquema general del establecimiento de claves mediante un KEM

460. El algoritmo ML-KEM considera tres conjuntos de parámetros, de forma análoga a como lo hace Kyber: ML-KEM-512 (categoría de seguridad 1), ML-KEM-768 (categoría de seguridad 3) y ML-KEM-1024 (categoría de seguridad 5).

461. Debe tenerse en cuenta que si las entradas a los diferentes algoritmos son las adecuadas, el procedimiento de establecimiento de claves de ML-KEM nunca fallará de forma explícita. De hecho, los algoritmos ML-KEM.Encaps y ML-KEM.Decaps siempre generarán un valor con el mismo tipo de datos que una clave secreta compartida y nunca generarán un símbolo de error o fallo. Sin embargo, es posible, aunque muy improbable, que el proceso falle de modo que Alicia (con ML-KEM.Decaps) y Bernardo (con ML-KEM.Encaps) produzcan resultados diferentes, aunque ambos se comporten honestamente y no haya interferencias de posibles adversarios. Si este hecho se produce, se dice que hay un fallo de desencapsulado.

462. Las estimaciones de probabilidad de fallo de desencapsulado para cada conjunto de parámetros de ML-KEM se muestran en la Tabla B.6.

Conjunto de parámetros	Fallo de desencapsulado
ML-KEM-512	$2^{-138,8}$
ML-KEM-768	$2^{-164,8}$
ML-KEM-1024	$2^{-174,8}$

Tabla B.6: Estimaciones de probabilidad de fallo de desencapsulado

463. Los algoritmos que define ML-KEM emplean subrutinas basadas en un esquema de cifrado de clave pública denominado K-PKE. Los algoritmos de generación de claves (K-PKE.KeyGen), cifrado (K-PKE.Encrypt) y descifrado (K-PKE.Decrypt) son solo componentes del ML-KEM, por lo que ninguno de ellos está aprobado para ser utilizado como esquema criptográfico independiente. La descripción detallada de estos algoritmos puede verse en [FIPS203, §5].

464. Al margen de los tres algoritmos anteriores, existen otros algoritmos auxiliares que son necesarios para la ejecución completa del ML-KEM, como BitsToBytes, BytesToBits, Compress, Decompress, ByteEncode, ByteDecode, SampleNTT, SamplePolyCBD, NTT, NTT^{-1}, MultiplyNTTs y BaseCaseMultiply (para una descripción pormenorizada de cada uno de ellos, véase [FIPS203, §4.2, §4.3]).

465. Para cada conjunto de parámetros de ML-KEM, se asignan valores numéricos específicos de los parámetros individuales n, q, k, η_1, η_2, d_u y d_v. Se sabe que hay dos parámetros fijos para todas las configuraciones: $n = 256$ y $q = 3329$; los restantes parámetros difieren para cada uno de los tres conjuntos de parámetros de ML-KEM.

466. Por otra parte, ML-KEM emplea, como es sabido, tres componentes de un PKE, conocido como K-PKE, que no está aprobado para su uso de manera independiente. Su uso es exclusivamente como una colección de subrutinas para ser empleada en los algoritmos del estándar ML-KEM. K-PKE está formado por tres algoritmos: Generación de claves, K-PKE.KeyGen, Cifrado, K-PKE.Encrypt, y Descifrado, K-PKE.Decrypt (los algoritmos correspondientes se presentan de forma precisa en [FIPS203, §5.1-§5.3]).

467. Cuando se hace uso del K-PKE, como parte del estándar ML-KEM, este hereda los parámetros seleccionados para ML-KEM. Como ya se ha mencionado, siempre se consideran $n = 256$ y $q = 3329$, pero los restantes parámetros varían según el conjunto de parámetros elegido para ML-KEM. Por otra parte, conviene señalar que los tres algoritmos de K-PKE mencionados no verifican las entradas correspondientes dado que solo son llamados como subrutinas de los algoritmos de ML-KEM, que son los que verifican las entradas, según cada caso.

468. A continuación, se describen los tres algoritmos fundamentales que constituyen el nuevo estándar ML-KEM: Generación de claves, ML-KEM.KeyGen, Encapsulado, ML-KEM.Encaps, y Desencapsulado, ML-KEM.Decaps (cada uno de los cuales se muestra en [FIPS203, §7.1-§7.3]). Estos tres algoritmos remiten en su ejecución a un algoritmo interno (denominados *main internal algorithms*), que, por completitud de esta Guía, serán incluidos en lo que sigue (puede verse una descripción más detallada de cada uno en [FIPS203, §6.1-§6.3]). Tales algoritmos se denotan de la misma forma que el algoritmo fundamental al que hacen referencia, pero incluyen la expresión "_internal" para ser distinguidos.

469. Los tres algoritmos internos son deterministas, es decir, su salida está completamente determinada por su entrada, sin que exista aleatoriedad dentro de ellos. Los parámetros que emplean son los correspondientes a los utilizados por algoritmos que les invocan.

470. Como ya hemos mencionado, para crear una instancia concreta de ML-KEM, se debe seleccionar un conjunto de parámetros específico, se debe garantizar que los tres algoritmos de ML-KEM solo se invocan con un conjunto de parámetros válido,

apropiado para la aplicación deseada, y que el mismo debe coincidir con el conjunto de parámetros asociado a las entradas proporcionadas de cada algoritmo.

471. El algoritmo de generación de claves interno, ML-KEM.KeyGen_internal (véase el Algoritmo 7) acepta dos semillas aleatorias como entrada y produce como salida una clave de encapsulado y otra de desencapsulado. Su subrutina principal es el algoritmo de generación de claves de K-PKE, K-PKE.KeyGen [FIPS203, §5.1], de modo que la clave de encapsulado es la clave de cifrado de K-PKE; mientras que la clave de desencapsulado está formada por la clave de descifrado de K-PKE, la clave de encapsulado, un hash de la clave de encapsulado y un valor aleatorio de 32 bytes. Este valor aleatorio se utiliza en el mecanismo de "rechazo implícito" del algoritmo de desencapsulado interno.

472. Se denota por $\mathbb{B}$ al conjunto $\{0, 1, \ldots, 255\}$ de los enteros sin signo de 8-bits y la función hash a emplear es siendo $H(\cdot) = \text{SHA3-256}(\cdot)$.

Algoritmo 7 Generación de claves interno: ML-KEM.KeyGen_internal(d,z)

Utiliza la aleatoriedad para generar una clave de encapsulado y una clave de desencapsulado asociada.
Entrada: un valor aleatorio $d \in \mathbb{B}^{32}$
Entrada: un valor aleatorio $z \in \mathbb{B}^{32}$
Salida: clave de encapsulado $ek \in \mathbb{B}^{32}$
Salida: clave de desencapsulado $dk \in \mathbb{B}^{32}$.

1. $(ek_{\text{PKE}}, dk_{\text{PKE}}) \leftarrow \text{K-PKE.KeyGen}(d)$
2. $ek \leftarrow ek_{\text{PKE}}$
3. $dk \leftarrow (dk_{\text{PKE}} || ek || H(ek) || z)$
4. **devuelve** (ek, dk)

473. El algoritmo de generación de claves ML-KEM.KeyGen (ver el Algoritmo 8) no acepta ninguna entrada, genera aleatoriedad internamente y produce una clave de encapsulado y una de desencapsulado. La primera de ellas puede hacerse pública; mientras que la clave de desencapsulado debe permanecer secreta.

474. La semilla (d, z) generada en los pasos 1 y 2 del algoritmo ML-KEM.KeyGen se puede almacenar para una expansión posterior utilizando ML-KEM.KeyGen_internal. Dado que como esta semilla se puede utilizar para calcular la clave de desencapsulado, la misma debe considerarse como confidencial y se debe tratar con las mismas salvaguardas que una clave de desencapsulado.

Algoritmo 8 Generación de claves: ML-KEM.KeyGen()

Genera una clave de encapsulado y la clave de desencapsulado correspondiente.
Salida: clave de encapsulado $ek \in \mathbb{B}^{384k+32}$
Salida: clave de desencapsulado $dk \in \mathbb{B}^{768k+96}$.

1. $d \xleftarrow{\$} \mathbb{B}^{32}$
2. $z \xleftarrow{\$} \mathbb{B}^{32}$
3. **if** $d ==$ NULL **or** $z ==$ NULL **then**
4. **devuelve** $\perp$
5. **end if**
6. $(ek_{\text{PKE}}, dk_{\text{PKE}}) \leftarrow$ ML-KEM.KeyGen_internal(d, z)
7. **devuelve** (ek, dk)

475. La generación de una clave segura depende de la claves que se hayan generado con el algoritmo ML-KEM.KeyGen. Si las claves no fueron generadas por su propietario, este puede, opcionalmente, realizar determinadas comprobaciones con el fin de detectar posibles corrupciones, aunque el algoritmo que se propone a continuación, Key pair check (ver Algoritmo 9), no garantiza que las mismas se hayan generado correctamente.

Algoritmo 9 Comprobación de la generación de claves: Key pair check

Comprobación de un par de candidatos a clave $(\bar{ek}, \bar{dk})$.

1. (Consistencia de la semilla) Si hay una semilla disponible, (d, z), ejecutar ML-KEM.KeyGen_internal(d,z) y verificar que la salida es igual a $(\bar{ek}, \bar{dk})$.
2. (Verificación de clave de encapsulado) Verificar $\bar{ek}$ como se señala en el Algoritmo 12.
3. (Verificación de clave de desencapsulado) Verificar $\bar{dk}$ como se especifica en el Algoritmo 15.
4. (Consistencia por pares) Realizar los siguientes pasos:
   a) Generar una matriz de 32 bytes aleatorios ejecutando $m \xleftarrow{\$} \mathbb{B}^{32}$.
   b) Ejecutar $(K, c) \xleftarrow{\$}$ ML-KEM.Encaps_internal$(\bar{ek}, m)$.
   c) Ejecutar $K' \xleftarrow{\$}$ ML-KEM.Decaps_internal$(\bar{dk}, c)$.
   d) Rechazar a menos que $K == K'$.

476. Al igual que sucede con el algoritmo de generación de claves, el algoritmo de encapsulado, ML-KEM.Encaps, utiliza un algoritmo de encapsulado interno, denominado ML-KEM.Encaps_internal (ver Algoritmo 10). Este algoritmo acepta como entrada una clave de encapsulado y una matriz de bytes aleatorios, y produce como salidas: un texto cifrado y una clave compartida.

477. La subrutina principal del algoritmo ML-KEM.Encaps_internal es el algoritmo de cifrado de K-PKE, que se utiliza para cifrar un valor aleatorio, m, en un texto cifrado, c. Una copia de la clave secreta compartida, K, y la aleatoriedad utilizada durante el cifrado se derivan de m y de la clave de encapsulado, ek, a través de la función hash H. De forma más específica, H se aplica H a ek y el resultado se concatena con m y luego se calcula su hash mediante G. Por último, el algoritmo proporciona como salida la clave secreta compartida K y el texto cifrado c.

Algoritmo 10 Encapsulado interno: ML-KEM.Encaps_internal(ek, m)

Utiliza la clave de encapsulado y la aleatoriedad para generar una clave y un texto cifrado asociado. *Entrada:* clave de encapsulado $ek \in \mathbb{B}^{384k+32}$.
Entrada: aleatoriedad $m \in \mathbb{B}^{32}$
Salida: clave secreta compartida $K \in \mathbb{B}^{32}$
Salida: texto cifrado $c \in \mathbb{B}^{32(d_u k+d_v)}$.

1. $(K, r) \leftarrow G(m||H(ek))$
2. $c \leftarrow$ K-PKE.Encrypt(ek, m, r)
3. **devuelve** (K, c)

478. El algoritmo de encapsulado ML-KEM.Encaps (ver el Algoritmo 11) acepta una clave de encapsulado como entrada, genera aleatoriedad internamente y genera como salida un texto cifrado y una clave compartida.

Algoritmo 11 Encapsulado: ML-KEM.Encaps(ek)

Utiliza la clave de encapsulado para generar una clave secreta compartida y un texto cifrado asociado.
Validación de la entrada: clave de encapsulado $ek \in \mathbb{B}^{384k+32}$.
Salida: clave secreta compartida $K \in \mathbb{B}^{32}$
Salida: texto cifrado $c \in \mathbb{B}^{32(d_u k+d_v)}$.

1. $m \xleftarrow{\$} \mathbb{B}^{32}$
2. **if** $m ==$ NULL **then**
3. **devuelve** $\perp$
4. **end if**
5. $(K, c) \leftarrow$ ML-KEM.Encaps_internal(ek, m)
6. **devuelve** (K, c)

479. El algoritmo ML-KEM.Encaps requiere, como se ha visto, una verificación de la entrada. Esta se lleva a cabo tal y como se especifica en el algoritmo Encapsulate key check (véase el Algoritmo 12).

Algoritmo 12 Comprobación de la clave de encapsulado: Encapsulate key check

Comprobación de una clave de encapsulado, $\bar{ek}$.

1. (Comprobación de tipo) Si $\bar{ek}$ no es una matriz de bytes de longitud $384k + 32$ para el valor de k especificado por el conjunto de parámetros correspondiente, entonces la comprobación de la entrada falla.

2. (Comprobación del módulo) Realizar la siguiente operación: Si test $\neq \bar{ek}[0 : 384k]$, entonces la comprobación de entrada falla. Esta comprobación asegura que los números enteros codificados en la clave pública estén en el rango válido de $[0, q - 1]$.

480. Si ambas comprobaciones son correctas, entonces ML-KEM.Encaps se puede ejecutar con la entrada $ek := \bar{ek}$. Es importante tener en cuenta que, al igual que en el caso del Algoritmo 9, este proceso de comprobación no garantiza que ek sea una salida producida correctamente por ML-KEM.KeyGen.

481. Por otra parte, el algoritmo ML-KEM.Encaps no debe ejecutarse con una clave de encapsulado que no haya sido verificada. Sin embargo, no es necesario que la parte que realiza la encapsulado realice la verificación de la clave de encapsulado ni con cada ejecución de ML-KEM.Encaps.

482. Tal y como ya se ha mencionado, el algoritmo de desencapsulado ML-KEM.Decaps (ver Algoritmo 14) también hace uso del correspondiente algoritmo interno, ML-KEM.Decaps_internal.

483. El algoritmo de desencapsulado interno ML-KEM.Decaps_internal (ver Algoritmo 13) analiza los componentes de la clave de desencapsulado dk del ML-KEM. Tales componentes son las claves de cifrado y descifrado para K-PKE, un valor hash h y un valor aleatorio z. La clave de descifrado de K-PKE se emplea para descifrar el texto cifrado de la entrada, c, y obtener un texto claro, m'. A continuación, el algoritmo de desencapsulado vuelve a cifrar (proceso de recifrado) m' y calcula una clave secreta compartida candidata K' tal y como se debería haber hecho en el algoritmo de encapsulado. En concreto, 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' y utilizando la aleatoriedad r'. Por último, el desencapsulado comprueba si el texto cifrado resultante c' coincide con el texto cifrado proporcionado c. Si no es así, el algoritmo realiza un "rechazo implícito", esto es, el valor de K' se cambia por un hash de c junto con el valor aleatorio z, almacenado en la clave secreta ML-KEM. En cualquier caso, el desencapsulado genera la clave secreta compartida resultante K'.

484. El indicador del rechazo implícito, calculado en el paso 9 cuando se comparan c y c', es un dato intermedio secreto. Este indicador se debe destruir antes de que

ML-KEM.Decaps_internal finalice. De hecho, no se permite devolver el valor del indicador como salida en ningún formato.

Algoritmo 13 Desencapsulado interno: ML-KEM.Decaps_internal(c, dk)

Utiliza la clave de desencapsulado para generar una clave secreta compartida a partir de un texto cifrado.
Entrada: clave de desencapsulado $dk \in \mathbb{B}^{768k+96}$
Entrada: texto cifrado $c \in \mathbb{B}^{32(d_u k+d_v)}$
Salida: clave secreta compartida $K \in \mathbb{B}^{32}$.

1. $dk_{\text{PKE}} \leftarrow dk[0 : 384k]$
2. $ek_{\text{PKE}} \leftarrow dk[384k : 768k + 32]$
3. $h \leftarrow dk[768k + 32 : 768k + 64]$
4. $z \leftarrow dk[768k + 64 : 768k + 96]$
5. $m' \leftarrow$ K-PKE.Decrypt(dk_{PKE}, c)
6. $(K', r') = G(m'||h)$
7. $\bar{K} \leftarrow J(z||c, 32)$
8. $c' \leftarrow$ K-PKE.Encrypt$(ek_{\text{PKE}}, m', r')$
9. **if** $c \neq c'$ **then**
10. $\quad K' = \bar{K}$
11. **end if**
12. **devuelve** K'

485. El algoritmo de desencapsulado, ML-KEM.Decaps (véase el Algoritmo 14) acepta una clave de desencapsulado y un texto cifrado de ML-KEM como entrada, no utiliza ningún tipo de aleatoriedad y genera un secreto compartido.

Algoritmo 14 Desencapsulado: ML-KEM.Decaps(dk, c)

Utiliza la clave de desencapsulado para generar una clave secreta compartida a partir de un texto cifrado.
Validación de la entrada: clave de desencapsulado $dk \in \mathbb{B}^{768k+96}$
Validación de la entrada: texto cifrado $c \in \mathbb{B}^{32(d_u k+d_v)}$
Salida: clave secreta compartida $K \in \mathbb{B}^{32}$.

1. $K' \leftarrow$ ML-KEM.Decaps_internal(dk, c)
2. **if** $c \neq c'$ **then**
3. **devuelve** K'

486. El algoritmo ML-KEM.Decaps, como los anteriores, requiere una comprobación de la entrada, tal y como se especifica en el algoritmo (véase el Algoritmo 15).

Algoritmo 15 Comprobación de la clave de encapsulado: Encapsulate key check

Comprobación de una clave de desencapsulado, $\bar{dk}$, y de un texto cifrado $\bar{c}$.

1. (Verificación del tipo de texto cifrado) Si $\bar{c}$ no es una matriz de bytes de longitud $32(d_u k + d_v)$ para los valores de d_u, d_v y k especificados por el conjunto de parámetros, entonces la verificación de entrada falla.
2. (Verificación del tipo de clave de desencapsulado) Si $\bar{dk}$ no es una matriz de bytes de longitud $768k + 96$ para el valor de k especificado por el conjunto de parámetros, entonces la verificación de entrada falla.
3. (Verificación de hash) Realizar el siguiente cálculo: test $\leftarrow H\left(\bar{dk}[384k : 768k + 32]\right)$.

   Si test $\neq dk[768k + 32 : 768k + 64]$, entonces la verificación de entrada falla.

487. Si las comprobaciones anteriores son correctas, se puede ejecutar ML-KEM.Decaps con las entradas $dk := \bar{dk}$ y $c := \bar{c}$. Se debe tener en cuenta, como ya se ha señalado, que este proceso de comprobación no garantiza que dk sea una salida producida correctamente por ML-KEM.KeyGen, ni que c sea una salida producida correctamente de ML-KEM.Encaps.

488. ML-KEM.Decaps no se debe ejecutar con una clave de desencapsulado o un texto cifrado a menos que ambos hayan sido comprobados. Sin embargo, la comprobación de la clave de desencapsulado no necesita ser realizada por la parte que desencapsula, ni con cada ejecución de ML-KEM.Decaps. La comprobación del texto cifrado se debe realizar con cada ejecución de ML-KEM.Decaps.

489. A la vista de los algoritmos anteriores, se puede afirmar que existen algunas diferencias entre CRYSTALS-Kyber y la versión de ML-KEM publicada por el NIST.

490. Las diferencias más destacables son las que se derivan de un comportamiento diferente en la entrada y la salida de los tres algoritmos principales: la generación de claves, el encapsulado y el desencapsulado. Esto es, no se detallan las diferencias de cómo estos algoritmos obtienen la salida a partir de la entrada. En todo caso, debe tenerse en cuenta que cualquier implementación que sea conforme con la propuesta con la del NIST debe coincidir con el comportamiento de las entradas y salidas de los tres algoritmos mencionados y de sus correspondientes algoritmos internos.

491. En particular, las principales diferencias son las siguientes:

- En la especificación de Kyber de la tercera ronda del NIST, la clave secreta compartida era un valor de longitud variable cuya longitud dependía de cómo se usaría esta clave en la aplicación que se considerara. En la versión de ML-KEM, la longitud de la clave secreta compartida se ha fijado en 256 bits. Ello permite que esta clave se pueda utilizar directamente en aplicaciones de clave simétrica, por ejemplo.
- Los algoritmos ML-KEM.Encaps y ML-KEM.Decaps utilizan una variante diferente de la transformada de Fujisaki-Okamoto que se emplea en la versión de Kyber de la tercera ronda. De hecho, ML-KEM.Encaps no incluye un hash del texto cifrado en la derivación del secreto compartido, por lo que ML-KEM.Decaps se ha modificado para que se ajuste con este cambio.
- En la versión de Kyber de la tercera ronda, la aleatoriedad inicial m en el algoritmo de encapsulado era primero modificada mediante un hash. El propósito de este paso era el de proteger el valor de m contra el uso de algoritmos de generación de datos aleatorios defectuosos. Sin embargo, como en ML-KEM se requiere el uso de generadores aleatorios aprobados por el NIST, este paso se considera innecesario y no se ejecuta en ML-KEM.
- En la especificación de ML-KEM, se lleva a cabo la verificación de las entradas, cosa que no se realizaba en la versión de Kyber de la tercera ronda. En todo caso, debe recordarse que esta verificación no se lleva a cabo en los correspondientes algoritmos internos.

492. Como ya hemos mencionado, ML-KEM tiene tres configuraciones diferentes para tres conjuntos de parámetros, cada uno de los cuales está formado por dos parámetros fijos, $n = 256$ y $q = 3329$, y otros cinco parámetros, k, η_1, η_2, d_u y d_v, cuyo valor varía para cada configuración.

493. La descripción de cada uno de los parámetros no fijos es la siguiente:

- El parámetro k determina las dimensiones de los vectores s y e en K-PKE.KeyGen, así como las dimensiones de la matriz A y los vectores r, e_1 y e_2 en K-PKE.Encrypt.
- El parámetro η_1 señala la distribución a utilizar para generar los vectores s y e en K-PKE.KeyGen y el vector r en K-PKE.Encrypt.
- El parámetro η_2 especifica la distribución para generar los vectores e_1 y e_2 en K-PKE.Encrypt.
- Los parámetros d_u y d_v sirven como parámetros y entradas para las funciones Compress, Decompress, ByteEncode y ByteDecode en K-PKE.Encrypt y K-PKE.Decrypt.

494. El estándar publicado por el NIST en [FIPS203] considera aprobados los conjuntos de parámetros que se muestran en la Tabla B.7.

Conjunto	n	q	k	η_1	η_2	d_u	d_v	Fortaleza RBG requerida
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 B.7: Conjunto de parámetros aprobados para ML-KEM

495. Por su parte, los tamaños de las claves (en bytes) para ML-KEM y los textos cifrados para cada conjunto de parámetros se presentan en la Tabla B.8.

Conjunto	Clave encapsulado	Clave desencapsulado	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 B.8: Tamaño en bytes de las claves y textos cifrados para ML-KEM

B.2.3. FRODOKEM

496. FrodoKEM es un KEM basado en un esquema de cifrado que fundamenta su seguridad en el problema LWE. Este algoritmo se distingue de otros algoritmos basados en retículos en que no utiliza una estructura de anillo o módulo, lo que hace que el algoritmo gane en seguridad, pero pierda en funcionalidad y que tenga longitudes de claves más grandes.

497. FrodoKEM está diseñado considerando una función hash que toma como entradas un texto claro elegido al azar y el hash de la clave pública, y genera una cadena de bits grande. El hecho de introducir la clave pública en la generación de r o K

no supone un cambio relevante desde el punto de vista de la seguridad y tiene la ventaja de proporcionar una seguridad multiobjetivo más fuerte [FRODO-KEM]. En relación con la seguridad de FrodoKEM, se ha demostrado [BCD+16, Th.5.1] que la seguridad IND-CCA del KEM resultante se reduce a la seguridad IND-CPA del PKE subyacente. La prueba de seguridad de FrodoKEM en QROM radica en los resultados de Jian et al. [JZC+17].

498. Los algoritmos de generación de claves, encapsulado y desencapsulado se muestran como Algoritmos 16, 17 y 18, respectivamente. Se puede apreciar que en este caso se emplea un doble hash que involucra a la clave pública para generar la entrada aleatoria del algoritmo de cifrado.

Algoritmo 16 Frodo: Generación de claves $\mathcal{G}$

1. $(pk, sk) \leftarrow \mathcal{G}'$
2. $s \leftarrow_R \{0,1\}^{len_s}$
3. $pkh = H_1(pk)$
4. $sk' = (sk, s, pk, pkh)$
5. **Devuelve** (pk, sk')

Algoritmo 17 Frodo: Encapsulado $\mathcal{E}c(pk)$

1. $m \leftarrow MSPC$
2. $(r, k) \leftarrow H_2(H_1(pk)||m)$
3. $c \leftarrow \mathcal{E}(pk, m; r)$
4. $K = H_3(c||k)$
5. **Devuelve** (c, K)

Algoritmo 18 Frodo: Desencapsulado $\mathcal{D}c(c, sk')$

1. $m' \leftarrow \mathcal{D}(sk, c)$
2. $(r', k') = H_2(pkh||m)$
3. $K'_0 = H_3(c||k')$
4. $K'_1 = H_3(c||s)$
5. **if** $c = \mathcal{E}(m', pk; r')$ **then** $K' = K'_0$
6. **else** $K' = K'_1$
7. **Devuelve** K'

B.2.4. ML-DSA

499. El NIST ha publicado en [FIPS204] un estándar de firma digital llamado *Module-Lattice-Based Digital Signature Standard* (ML-DSA). Por esta razón, en esta sección presentaremos las características generales de este nuevo estándar, que se basa en los mismos principios que CRYSTALS-Dilithium.

500. ML-DSA consta de tres algoritmos: generación de claves, ML-DSA-KeyGen (Algoritmo 20), elaboración de la firma, ML-DSA-Sign (Algoritmo 22), y verificación de la misma, ML-DSA-Verify (Algoritmo 22) y se ha diseñado para cumplir con la definición de seguridad SUF-CMA (*Strong Existential UnForgeability under Chosen Message Attack*). Esto es, se espera que incluso si un adversario pudiera lograr que una parte honesta firme mensajes arbitrarios, dicho adversario no podría elaborar ninguna firma válida adicional basada en la clave pública del firmante, incluyendo en el caso de mensajes ya firmado por el firmante. ML-DSA también está diseñado para satisfacer otras propiedades de seguridad adicionales como las que se presentan en [CDF+21].

501. Además de estos tres algoritmos, existen otros algoritmos auxiliares que son necesarios para la ejecución completa del ML-DSA. Estos son: IntegerToBits, BitsToInteger, IntegerToBytes, BitsToBytes, BytesToBits, CoeffFromThreeBytes, CoeffFromHalfByte, SimpleBitPack, BitPack, SimpleBitUnpack, BitUnpack, HintBitPack, HintBitUnpack, pkEncode, pkDecode, skEncode, skDecode, sigEncode, sigDecode, w1Encode, SampleInBall, RejNTTPoly, RejBoundedPoly, ExpandA, ExpandS, ExpandMask, Power2Round, Decompose, HighBits, LowBits, MakeHint, UseHint, NTT, NTT-1, BitRev_8, AddNTT, MultiplyNTT, AddVectorNTT, ScalarVectorNTT y MatrixVectorNTT (para una descripción pormenorizada de cada uno de ellos, véase [FIPS203, §7.2-§7.6]).

502. Por otra parte, en [FIPS204] se define un nuevo esquema de firma que está estrechamente relacionado con ML-DSA, conocido como HashML-DSA. Este se

diferencia del primero en el hecho de que incluye un paso de prehash adicional antes de la firma. Al igual que el ML-DSA, HashML-DSA también consta de tres algoritmos: el de generación de claves, que es el mismo algoritmo utilizado para ML-DSA, ML-DSA.KeyGen (Algoritmo 20), el de firma, HashML-DSA.Sign (Algoritmo 25), y el de verificación, HashML-DSA.Verify (Algoritmo 26).

503. La construcción básica de ML-DSA es similar a los esquemas de firma basados en retículos, es decir, se trata de construir un esquema de firma a partir de un protocolo interactivo donde un probador que conoce las matrices $A \in \mathbb{Z}_q^{K\times L}$, $S_1 \in \mathbb{Z}_q^{L\times n}$ y $S_2 \in \mathbb{Z}_q^{K\times n}$ con coeficientes pequeños, demuestra el conocimiento de tales matrices a un verificador que conoce A y $T = AS_1 + S_2 \in \mathbb{Z}_q^{K\times n}$. De forma general, un protocolo interactivo de este tipo seria como el siguiente:

    1. Compromiso: el probador genera $y \in \mathbb{Z}_q^L$ con coeficientes pequeños y se compromete con su valor enviando $w_{Approx} = Ay + y_2$ al verificador, siendo $y_2 \in \mathbb{Z}_q^K$ un vector con coeficientes pequeños.

    2. Desafío: el verificador envía un vector $c \in \mathbb{Z}_q^n$ con coeficientes pequeños al probador.

    3. Respuesta: El probador devuelve $z = y + S_1 c$ y el verificador comprueba que z tiene coeficientes pequeños y que $Az - Tc \approx w_{Approx}$.

504. Este protocolo presenta un fallo de seguridad dado que la respuesta z está sesgada según el valor privado S_1. Del mismo modo, $r = w_{Approx} - Az + Tc = y_2 + S_2 c$ está sesgado por el valor privado S_2. Sin embargo, este defecto se puede corregir cuando el protocolo interactivo anterior se convierte en un esquema de firma. Esto es, al igual que con las firmas de Schnorr, el firmante deriva el desafío mediante un proceso pseudoaleatorio a partir de un hash del compromiso concatenado con el mensaje. Sin embargo, para corregir el sesgo, el firmante aplica un muestreo de rechazo a z de modo que si los coeficientes de z caen fuera de un determinado rango, el proceso de firma se cancela y el firmante comienza otra vez con un nuevo valor de y. También se debe aplicar un muestreo de rechazo similar a r. En la firma resultante, de tipo Fiat-Shamir with aborts, la clave pública es (A, T) y la clave privada es (S_1, S_2).

505. En cualquier caso, en el estándar ML-DSA se han añadido determinados ajustes y modificaciones al protocolo básico anterior por razones de seguridad y eficiencia. Estos son:

    - Para reducir el tamaño de la clave y de la firma y utilizar una multiplicación polinómica rápida basada en NTT, ML-DSA utiliza matrices estructuradas por módulos. Es decir, reemplaza bloques de matrices de dimensión $n \times n$ y bloques de vectores de dimensión n por polinomios en el anillo R_q. Por tanto, en lugar de $A \in \mathbb{Z}_q^{K\times L}$, $S_1 \in \mathbb{Z}_q^{L\times n}$, $S_2 \in \mathbb{Z}_q^{K\times n}$, $T = AS_1 + S_2 \in \mathbb{Z}_q^{K\times n}$,

$y \in \mathbb{Z}_q^L$ y $c \in \mathbb{Z}_q^n$, ML-DSA considera $A \in R_q^{k \times l}$, $t \in R_q^k$, $s_1 \in R_q^l$, $s_2 \in R_q^k$, $y \in R_q^l$ y $c \in R_q$, donde $l = L/n$ y $k = K/n$.

- Para reducir aún más el tamaño de la clave pública, la matriz A se deriva pseudoaleatoriamente de una semilla pública de 256 bits, ρ, que se incluye en la clave pública ML-DSA en lugar de A.
- Para reducir todavía más el tamaño de la clave pública, ML-DSA sustituye t por un valor comprimido t_1, que elimina los d bits de orden inferior de cada coeficiente.
- Para obtener propiedades que eviten la falsificación (BUFF), ML-DSA no firma el mensaje M directamente, sino que firma un mensaje representativo μ que se obtiene mediante hash de la concatenación de un hash de la clave pública y M.
- Para reducir el tamaño de la firma, en lugar de incluir el compromiso $w_{Approx} = Ay + y_2$ en la firma, ML-DSA utiliza una versión redondeada $w = Ay$ como compromiso w_1 e incluye solo el hash $\tilde{c}$ de $w_1||\mu$.
- Para garantizar que el verificador pueda reconstruir w_1 a partir de z y el valor comprimido t_1, la firma también debe incluir una pista $h \in R_2^k$ calculada por el firmante utilizando su clave privada.
- Además, para garantizar la corrección, se debe incluir una segunda etapa de muestreo de rechazo (ver línea 25 del Algoritmo 21).

506. Al margen de otras recomendaciones y requisitos adicionales que el NIST señala en su publicación [FIPS204, §3.4-§3.6], merece especial atención el uso que este estándar hace de las funciones hash SHAKE256 and SHAKE128 [FIPS202]. De hecho, si bien el FIPS 202 especifica que estas funciones reciben y generan cadenas de bits, la mayoría de las implementaciones tratan las entradas y las salidas como cadenas de bytes. En este estándar de firma digital, siempre se llamará a estas funciones con una longitud de salida que será un múltiplo de ocho bits y tratará la salida de las mismas como una cadena de bytes, que será la misma cadena de bytes que resultaría de tomar la cadena de bits esperada de una lectura literal de FIPS 202 y procesarla con la función BitsToBytes. Sin embargo, con el fin de permitir que se puedan firmar mensajes que no tengan un número entero de bytes, el estándar de ML-DSA sobrecargará SHAKE256 de modo que su entrada pueda ser una cadena de bits, aunque generalmente será una cadena de bytes. La siguiente equivalencia se mantendrá para cualquier cadena de bytes str y el entero $l \geq 1$:

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

507. Por otra parte, ML-DSA utiliza, en ocasiones, la API incremental definida en [SP800-185], que consta de tres funciones para cada variante de SHAKE. Tales

funciones se pueden utilizar para inicializar (Init) una función hash, absorber (Absorb) una secuencia de cadenas de longitud arbitraria o comprimir (Squeeze) una secuencia de cadenas de longitud arbitraria. De forma resumida, si H y G denotan, respectivamente SHAKE256 y SHAKE128, también se hará uso ocasional de las funciones $H.Init$, $G.Init$, $H.Absorb$, $G.Absorb$, $H.Squeeze$ y $G.Squeeze$.

508. Además de SHAKE128 y SHAKE256, los algoritmos HashML-DSA.Sign y HashML-DSA.Verify pueden llamar a otras funciones hash que estén aprobadas para el hash previo. El pseudocódigo de este estándar también trata estas funciones como si devolvieran una cadena de bytes como salida, mientras que admiten como entrada una cadena de bits o una cadena de bytes.

509. A continuación se presentan los tres algoritmos principales, ya mencionados y que pueden llamarse externos, que forman parte del estándar ML-DSA. Para su ejecución, se recurre a otros algoritmos, llamados internos a los que se añade, como en el caso de ML-KEM, el sufijo "_internal". Por completitud, se presentarán, en lo que sigue, tanto los internos como los externos para cada uno de los procesos: generación de claves, firma y verificación. Posteriormente también se incluirán los algoritmos relativos al prehash adicional, esto es, los HashML-DSA.

510. El algoritmo interno de generación de claves, ML-DSA.KeyGen_internal (ver Algoritmo 19) toma una semilla aleatoria de 32 bytes como entrada y genera una clave pública y una clave privada, ambas codificadas como cadenas de bytes.

Algoritmo 19 Generación de claves interno: ML-DSA.KeyGen_internal(ξ)

Entrada: semilla $\xi \in \mathbb{B}^{32}$
Salida: clave pública $pk \in \mathbb{B}^{32+32k(\text{bitlen}(q-1)-d)}$ y una clave privada $sk \in \mathbb{B}^{32+32+64+32((l+k)\text{bitlen}(2\eta)+dk)}$.

1. $(\rho, \rho', K) \in \mathbb{B}^{32} \times \mathbb{B}^{64} \times \mathbb{B}^{32} \leftarrow H(\xi||IntegerToBytes(k,1)||IntegerToBytes(l,1),128)$
2. $\hat{A} \leftarrow \text{ExpandA}(\rho)$
3. $(s_1, s_2) \leftarrow \text{ExpandS}(\rho')$
4. $t \leftarrow \text{NTT}^{-1}(\hat{A} \circ \text{NTT}(s_1)) + s_2$
5. $(t_1, t_0) \leftarrow \text{Power2Round}(t, d)$
6. $pk \leftarrow \text{pkEncode}(\rho, t_1)$
7. $tr \leftarrow H(pk, 64)$
8. $sk \leftarrow \text{skEncode}(\rho, K, tr, s_1, s_2, t_0)$
9. **devuelve** (pk, sk)

511. Por su parte, el algoritmo de generación de claves, ML-DSA.KeyGen (ver Algoritmo 20) no recibe ninguna entrada y genera una clave pública y una clave privada, ambas codificadas como cadenas de bytes. El algoritmo utiliza un RBG aprobado para generar una semilla aleatoria de 256 bits (32 bytes) que se proporciona como entrada a ML-DSA.KeyGen_internal y que genera las claves pública y privada.

Algoritmo 20 Generación de claves: ML-DSA.KeyGen()

Genera un par de claves pública y privada. *Salida:* clave pública $pk \in \mathbb{B}^{32+32k(\text{bitlen}(q-1)-d)}$ y una clave privada $sk \in \mathbb{B}^{32+32+64+32((l+k)\text{bitlen}(2\eta)+dk)}$.

1. $\xi \xleftarrow{\$} \mathbb{B}^{32}$
2. **if** $\xi ==$ NULL **then**
3. **devuelve** $\perp$
4. **end if**
5. **devuelve** ML-DSA.KeyGen_internal(ξ)

512. El algoritmo interno para la elaboración de la firma, ML-DSA.Sign_internal (ver Algoritmo 21) considera como entrada una clave privada sk, codificado como una cadena de bytes, un mensaje formateado, M', codificado como una cadena de bits, y una cada de 32 bytes, rnd, y proporciona como salida una firma para el mensaje codificada como una cadena de bytes.

513. Hay dos formas en las que un algoritmo de firma puede usar ML-DSA.Sign_internal: la protegida y la determinista. Las variantes protegidas predeterminadas de ML-DSA.Sign y HashML-DSA.Sign utilizan un valor aleatorio nuevo para rnd, mientras que las variantes deterministas opcionales utilizan la cadena de bytes constante $\{0\}^{32}$.

514. En las dos variantes mencionadas, el firmante extrae de la clave privada: la semilla aleatoria pública ρ; la semilla aleatoria privada de 32 bytes, K; el hash de 64 bytes de la clave pública, tr, los vectores polinómicos secretos s_1 y s_2 y el vector polinómico t_0, que codifica los d bits menos significativos de cada coeficiente del polinomio de clave pública sin comprimir t. Luego, ρ se expande a la misma matriz A que en la generación de claves.

515. Antes de firmar el mensaje, M, este se concatena con el hash de clave pública tr y se reduce a un representante del mensaje de 64 bytes, μ, utilizando la función hash H.

516. El firmante produce entonces una semilla adicional de 64 bytes, $rho'' \leftarrow H(K||rnd||\mu, 64)$, para obtener una aleatoriedad privada durante cada operación de firma. En la variante protegida predeterminada, rnd es la salida de un RBG;

mientras que en la variante determinista rnd es una cadena de 32 bytes que consta enteramente de ceros.

517. La parte principal del algoritmo interno de firma consiste en un bucle de muestreo de rechazo, donde cada iteración del bucle produce una firma válida o una firma no válida cuya publicación filtraría información sobre la clave privada. El bucle se repite hasta que se produce una firma válida, que luego se codifica como una cadena de bytes yes la salida. El ciclo de muestreo de rechazo sigue el paradigma de las firmas de Fiat-Shamir with aborts y (aparte del paso de rechazo) es similar en estructura a las firmas Schnorr (por ejemplo, la EdDSA, ver §3.2.2). El firmante produce primero un compromiso w_1, luego determina pseudoaleatoriamente un desafío c de w_1 y el mensaje representativo μ. Finalmente, el firmante calcula una respuesta z.

Algoritmo 21 Elaboración de la firma digital interno: ML-DSA.Sign_internal

Algoritmo determinista para generar una firma para un mensaje formateado M'.
Entrada: clave privada $sk \in \mathbb{B}^{32+32+64+32((l+k)\text{bitlen}(2\eta)+dk)}$, el mensaje a firmar formateado $M \in \{0,1\}^*$ y aleatoriedad o variable ficticia $rnd \in \mathbb{B}^{32}$
Salida: firma $\sigma \in \mathbb{B}^{\lambda/4+32l(1+\text{bitlen}(\gamma_1-1)+\omega+k)}$

1. $(\rho, K, tr, s_1, s_2, t_0) \leftarrow \text{skDecode}(sk)$
2. $\hat{s}_1 \leftarrow \text{NTT}(s_1)$
3. $\hat{s}_2 \leftarrow \text{NTT}(s_2)$
4. $\hat{t}_0 \leftarrow \text{NTT}(t_0)$
5. $\hat{A} \leftarrow \text{ExpandA}(\rho)$
6. $\mu \leftarrow H(\text{BytesToBits}(tr)||M', 64)$
7. $\rho'' \leftarrow H(K||rnd||\mu, 64)$
8. $\kappa \leftarrow 0$
9. $(z, h) \leftarrow \perp$
10. **while** $(z, h) = \perp$ **do**
11. $\quad y \in R_q^l \leftarrow \text{ExpandMask}(\rho'', \kappa)$
12. $\quad w \leftarrow \text{NTT}^{-1}(\hat{A} \circ \text{NTT}(y))$
13. $\quad w_1 \leftarrow HighBits(w)$
14. $\quad \tilde{c} \leftarrow H(\mu||\text{w1Encode}(w_1), \lambda/4)$
15. $\quad c \in R_q \leftarrow \text{SampleInBall}(\tilde{c})$
16. $\quad \hat{c} \leftarrow \text{NTT}(c)$
17. $\quad \ll cs_1 \gg \leftarrow \text{NTT}^{-1}(\hat{c} \circ \hat{s}_1)$
18. $\quad \ll cs_2 \gg \leftarrow \text{NTT}^{-1}(\hat{c} \circ \hat{s}_2)$
19. $\quad z \leftarrow y + \ll cs_1 \gg$
20. $\quad r_0 \leftarrow \text{LowBits}(w - \ll cs_2 \gg)$
21. $\quad$ **if** $||z||_\infty \geq \gamma_1 - \beta$ **or** $||r_0||_\infty \geq \gamma_2 - \beta$ **then** $(z, h) \leftarrow \perp$
22. $\quad$ **else**
23. $\quad\quad \ll ct_0 \gg \leftarrow \text{NTT}^{-1}(\hat{c} \circ \hat{t}_0)$
24. $\quad\quad h \leftarrow \text{MakeHint}(- \ll ct_0 \gg, w - \ll cs_2 \gg + \ll ct_0 \gg)$
25. $\quad\quad$ **if** $|| \ll ct_0 \gg ||_\infty \geq \gamma_2$ **or** # 1 en $h \geq \omega$ **then** $(z, h) \leftarrow \perp$
26. $\quad\quad$ **end if**
27. $\quad$ **end if**
28. $\quad \kappa \leftarrow \kappa + l$
29. **end while**
30. $\sigma \leftarrow \text{sigEncode}(\tilde{c}, z \bmod^{\pm} q, h)$
31. **devuelve** σ

518. El algoritmo para la elaboración de la firma, ML-DSA.Sign (ver Algoritmo 22) considera como entrada una clave privada, un mensaje y una cadena de contexto y proporciona como salida una firma que está codificada como una cadena de bytes. Para la versión protegida predeterminada de la firma ML-DSA, el algoritmo utiliza un RBG aprobado para generar una semilla aleatoria de 32 bytes, rnd. En el caso de la variante determinista, entonces rnd se establece como la cadena fija $\{0\}^{32}$. Los valores rnd, la clave privada y el mensaje codificado se toman como entrada para ML-DSA.Sign_internal (Algoritmo 21), que es quien produce la firma.

Algoritmo 22 Elaboración de la firma digital: ML-DSA.Sign

Genera una firma ML-DSA.
Entrada: clave privada $sk \in \mathbb{B}^{32+32+64+32((l+k)\text{bitlen}(2\eta)+dk)}$, mensaje a firmar $M \in \{0,1\}^*$ y cadena de contexto ctx (una cadena de 255 bytes o menos).
Salida: la firma $\sigma \in \mathbb{B}^{\lambda/4+32l(1+\text{bitlen}(\gamma_1-1)+\omega+k)}$

1. **if** $|ctr| > 255$ **then**
2. **devuelve** $\perp$
3. **end if**
4. $rnd \leftarrow \mathbb{B}^{32}$
5. **if** $rnd ==$ NULL **then**
6. **devuelve** $\perp$
7. **end if**
8. $M' \leftarrow$ BytesToBits(IntegerToBytes$(0,1)||$IntegerToBytes$(|ctx|,1)||ctx)||M$
9. $\sigma \leftarrow$ ML-DSA.Sign_internal(sk, M', rnd)
10. **devuelve** σ

519. El algoritmo interno de verificación de la firma digital, ML-DSA.Verify_internal (ver Algoritmo 23) toma como entrada una clave pública, pk, codificada como una cadena de bytes, un mensaje, M, codificado como una cadena de bits y una firma, σ, codificada como una cadena de bytes, y proporciona como salida un valor booleano de verdadero (si la firma es válida con respecto al mensaje y la clave pública) o falso (en caso contrario). Este algoritmo no requiere aleatoriedad y especifica las longitudes de la firma y la clave pública según los términos de los parámetros descritos en la Tabla B.9. Si una implementación de ML-DSA.Verify_internal puede aceptar entradas para σ o pk de cualquier otra longitud, devolverá falso siempre que la longitud de cualquiera de estas entradas difiera de su longitud especificada.

520. El verificador de la firma extrae la semilla aleatoria pública ρ y el vector comprimido t_1 de la clave pública, pk, y de la firma, σ, obtiene el hash del compromiso del firmante, $\tilde{c}$, la respuesta z y la pista h. El verificador puede comprobar si la pista no está codificada en bytes correctamente, lo que se indica con el símbolo $\perp$, en cuyo caso el algoritmo de verificación devolverá inmediatamente falso ($\perp$) para señalar que la firma no es válida.

521. En el caso de que los valores se extraigan correctamente, el verificador obtiene pseudoaleatoriamente A de ρ, tal y como se hace en la generación y firma de claves, y crea un mensaje representativo μ mediante el hash de la concatenación de tr (el hash de la clave pública pk) y el mensaje M. A continuación, el verificador intenta reconstruir el compromiso del firmante (el vector polinómico w_1) a partir de la clave pública pk y la firma σ. En ML-DSA.Sign_internal, w_1 se calcula redondeando $w = Ay$. En ML-DSA.Verify_internal, el valor reconstruido de w_1 se denomina w'_1, ya que puede haber sido calculado de otra manera si la firma no es válida.

522. Por último, el verificador comprueba que la respuesta z y la pista h del firmante son válidas, y que el valor de w'_1 es consistente con el hash del compromiso del firmante, $\tilde{c}$. De forma más precisa, el verificador comprueba que todos los coeficientes de z son lo suficientemente pequeños, es decir, están en el rango $(-(\gamma_1-\beta), \gamma_1-\beta)$, que h no contiene más de ω coeficientes distintos de cero y que $\tilde{c}$ coincide con el hash, $\tilde{c}'$, del mensaje representativo μ concatenado con w'_1. Si todas estas comprobaciones son correctas, el algoritmo devuelve verdadero; en caso contrario devuelve falso.

Algoritmo 23 Verificación de la firma digital interno: ML-DSA.Verify_internal(pk, M', σ)

Función interna para verificar una firma σ para un mensaje formateado M'.
Entrada: clave pública $pk \in \mathbb{B}^{32+32k(\text{bitlen}(q-1)-d)}$ y mensaje $M' \in \{0,1\}^*$
Entrada: firma $\sigma \in \mathbb{B}^{\lambda/4+32l(1+\text{bitlen}(\gamma_1-1))+\omega+k}$. *Salida:* valor booleano

1. $(\rho, t_1) \leftarrow \text{pkDecode}(pk)$
2. $(\tilde{c}, z, h) \leftarrow \text{sigDecode}(\sigma)$
3. **if** $h = \perp$ **then Devuelve** falso
4. **end if**
5. $\hat{A} \leftarrow \text{ExpandA}(\rho)$
6. $tr \leftarrow H(pk, 64)$
7. $\mu \leftarrow H(\text{BytesToBits}(tr)||M', 64)$
8. $c \in R_q \leftarrow \text{SampleInBall}(\tilde{c})$
9. $w'_{Approx} \leftarrow \text{NTT}^{-1}(\hat{A} \circ \text{NTT}(z) - \text{NTT}(c) \circ \text{NTT}(t_1 2^d))$
10. $w'_1 \leftarrow \text{UseHint}(h, w'_{Approx})$
11. $\tilde{c}' \leftarrow H(\mu||\text{w1Encode}(w'_1), lambda/4)$
12. **Devuelve** $[[||z||_\infty < \gamma_1 - \beta]]$ **and** $[[\tilde{c} = \tilde{c}']]$

523. El algoritmo de verificación de la firma digital, ML-DSA.Verify (ver Algoritmo 24) toma como entrada una clave pública pk codificada como una cadena de bytes, un mensaje M codificado como una cadena de bits, una firma s codificada como una cadena de bytes y una cadena de contexto, también codificada como una cadena de bytes. El algoritmo proporciona como salida un valor booleano de verdadero (si la firma es válida con respecto al mensaje y la clave pública) o falso (en caso contrario). La verificación se logra llamando a ML-DSA.Verify_internal (Algoritmo 23) con la clave pública, el mensaje codificado y la firma.

Algoritmo 24 Verificación de la firma digital: ML-DSA.Verify(pk, M, σ, ctx)

Verifica una firma σ para un mensaje M.
Entrada: clave pública $pk \in \mathbb{B}^{32+32k(\text{bitlen}(q-1)-d)}$, mensaje $M \in \{0,1\}^*$ y una firma $\sigma \in \mathbb{B}^{\lambda/4+32l(1+\text{bitlen}(\gamma_1-1))+\omega+k}$
Salida: valor booleano

1. **if** $|ctr| > 255$ **then**
2. **devuelve** $\perp$
3. **end if**
4. $M' \leftarrow$ BytesToBits(IntegerToBytes$(0,1)||$IntegerToBytes$(|ctx|,1)||ctx)||M$
5. **Devuelve** ML-DSA.Verify_internal(pk, M', σ)

524. Es importante señalar que para algunos módulos criptográficos que generan firmas de tipo ML-DSA, puede suceder que su rendimiento sea inaceptable si el mensaje M es grande, como podría pasar en el paso 6 del algoritmo ML-DSA.Sign_internal (Algoritmo 21). Para estos casos se propone el uso de algoritmos modificados, denominados genéricamente HashML-DSA, si bien, se recomienda el uso de los algoritmos ML-DSA siempre que sea posible.

525. El algoritmo de generación de claves para HashML-DSA es el mismo que el de ML-DSA (véase Algoritmo 20), pero no sucede lo mismo para los algoritmos de firma y verificación.

526. Con el fin de mantener el mismo nivel de seguridad cuando se utiliza HashML-DSA, el resumen que se firma debe generarse utilizando una función hash aprobada o XOF [FIPS180-4] o [FIPS202] que proporcione al menos λ bits de seguridad clásica contra ataques de colisión y de segunda preimagen. La verificación de una firma creada de esta manera requiere que la función de verificación genere un resumen del mensaje de la misma manera que se utilizará como entrada para la función de verificación.

527. En la versión HashML-DSA de firma (véase el Algoritmo 25), la entrada de mensaje a ML-DSA.Sign_internal es el resultado de aplicar una función hash o un XOF al contenido que se va a firmar. La salida del hash o XOF se antepone a un separador de dominio de un byte. El separador de dominio tiene un valor de uno para la firma prehash.

Algoritmo 25 Elaboración de la firma digital prehash: HashML-DSA.Sign(sk, M, ctx, PH)

Genera una firma ML-DSA prehash.
Entrada: clave privada $sk \in \mathbb{B}^{32+32+64+32((l+k)\text{bitlen}(2\eta)+dk)}$, mensaje a firmar $M \in \{0,1\}^*$, cadena de contexto ctx (una cadena de 255 bytes o menos) y una función prehash PH.
Salida: firma $\sigma \in \mathbb{B}^{\lambda/4+32l(1+\text{bitlen}(\gamma_1-1)+\omega+k)}$

1. **if** $|ctr| > 255$ **then**
2. **devuelve** $\perp$
3. **end if**
4. $rnd \leftarrow \mathbb{B}^{32}$
5. **if** $rnd ==$ NULL **then**
6. **devuelve** $\perp$
7. **end if**
8. **switch** PH **do**
9. **case** SHA-256
10. OID $\leftarrow$ IntegerToBytes$(0x0609608648016503040201, 11)$
11. $\text{PH}_M \leftarrow$ SHA256(M)
12. **case** SHA-512
13. OID $\leftarrow$ IntegerToBytes$(0x0609608648016503040203, 11)$
14. $\text{PH}_M \leftarrow$ SHA512(M)
15. **case** SHAKE-128
16. OID $\leftarrow$ IntegerToBytes$(0x060960864801650304020B, 11)$
17. $\text{PH}_M \leftarrow$ SHAKE128(M)
18. **case** ...
19. ...
20. **end switch**
21. $M' \leftarrow$ BytesToBits(IntegerToBytes$(1,1)$||IntegerToBytes$(|ctx|,1)$||ctx||OID||PH_M)
22. $\sigma \leftarrow$ ML-DSA.Sign_internal(sk, M', rnd)
23. **devuelve** σ

528. El Algoritmo 26 presenta la verificación de firma para HashML-DSA. En esta caso, M' se construye del mismo modo que en el Algoritmo 25 y pasa el M' resultante al algoritmo ML-DSA.Verify_internal para su verificación. Al igual que con la generación de firma previa al hash, M' puede construirse fuera del módulo criptográfico que realiza ML-DSA.Verify_internal. Sin embargo, en el caso de HashML-DSA, el hash o XOF del contenido debe calcularse dentro de un módulo criptográfico validado en [FIPS140-3], que puede ser un módulo criptográfico diferente al que realiza ML-DSA.Verify_internal.

Algoritmo 26 Verificación de la firma digital prehash: HashML-DSA.Verify$(pk, M, \sigma, ctx, \text{PH})$

Verifica una firma ML-DSA prehash.
Entrada: clave pública $pk \in \mathbb{B}^{32+32k(\text{bitlen}(q-1)-d)}$, mensaje $M \in \{0,1\}^*$, firma $\sigma \in \mathbb{B}^{\lambda/4+32l(1+\text{bitlen}(\gamma_1-1))+\omega+k}$, cadena de contexto ctx (una cadena de 255 bytes o menos) y una función prehash PH.
Salida: valor booleano

1. **if** $|ctr| > 255$ **then**
2. **devuelve** falso
3. **end if**
4. **switch** PH **do**
5. **case** SHA-256
6. OID $\leftarrow$ IntegerToBytes$(0x0609608648016503040201, 11)$
7. $\text{PH}_M \leftarrow$ SHA256(M)
8. **case** SHA-512
9. OID $\leftarrow$ IntegerToBytes$(0x0609608648016503040203, 11)$
10. $\text{PH}_M \leftarrow$ SHA512(M)
11. **case** SHAKE-128
12. OID $\leftarrow$ IntegerToBytes$(0x060960864801650304020B, 11)$
13. $\text{PH}_M \leftarrow$ SHAKE128(M)
14. **case** ...
15. ...
16. **end switch**
17. $M' \leftarrow$ BytesToBits(IntegerToBytes$(1,1)$||IntegerToBytes$(|ctx|,1)$||ctx||OID||PH_M)
18. **devuelve** ML-DSA.Verify_internal(pk, M', σ)

529. El NIST considera tres especificaciones diferentes para ML-DSA, denotadas como ML-DSA-44, ML-DSA-65 y ML-DSA-87, de modo que los dos últimos dígitos de

cada una de ellas hacen referencia al tamaño de la matriz A, esto es, ML-DSA-XY señala que el tamaño de A es $X \times Y$ sobre R_q.

530. El conjunto de parámetros establecido por el NIST para este esquema de firma estándar se presenta en [FIPS204] y es el de la Tabla B.9, cuyo significado se presenta a continuación.

Parámetros	ML-DSA-44	ML-DSA-65	ML-DSA-87
q	8380417	8380417	8380417
ζ	1753	1753	1753
d	13	13	13
τ	39	49	60
λ	128	192	256
γ_1	2^{17}	2^{19}	2^{19}
γ_2	$(q-1)/88$	$(q-1)/32$	$(q-1)/32$
(k,l)	$(4,4)$	$(6,5)$	$(8,7)$
η	2	4	2
$\beta = \tau \cdot \eta$	78	196	120
ω	80	55	75
Desafío de entropía $\log\binom{256}{\tau} + \tau$	192	225	257
Repeticiones	4,25	5,1	3,85
Fortaleza de la seguridad	Categoría 2	Categoría 3	Categoría 5

Tabla B.9: Conjunto de parámetros para ML-DSA

- ζ: una raíz 512-ésima de la unidad en $\mathbb{Z}_q$
- d: # de bits descartados de t
- τ: # de ± 1 del polinomio c
- λ: fortaleza de la colisión de $\tilde{c}$
- γ_1: rango de los coeficientes de y
- γ_2: rango de redondeo de orden inferior
- (k,l): dimensiones de A
- η: rango de la clave privada
- ω: máximo número de unos en la pista h
- Desafío de entropía $\log\left(\binom{256}{\tau}\right) + \tau$
- Repeticiones: número esperado de repeticiones del bucle principal en el algoritmo de firma

531. Tal y como se muestra en la Tabla B.9, la fortaleza de la seguridad de DL-DSA no se establece en términos de un número (128 bits de seguridad) sino que se afirma que cada conjunto de parámetros para ML-DSA es al menos tan seguro como un cifrado de bloque genérico con un tamaño de clave determinado o una función hash genérica con una longitud de salida establecida. Dicho de otro modo, se afirma que los recursos computacionales necesarios para romper ML-DSA son mayores o iguales que los recursos computacionales necesarios para romper el cifrado de bloque o la función hash considerada cuando estos recursos computacionales se estiman utilizando cualquier modelo realista de computación. En la Tabla B.10 se recuerdan las diferentes categorías de seguridad establecidas por el NIST.

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

Tabla B.10: Categorías de seguridad establecidas por el NIST

532. Finalmente, en la Tabla B.11 se muestran los tamaños (en bytes) de claves y de las firmas de ML-DSA.

Especificación	Clave privada	Clave pública	Tamaño de la firma
ML-DSA-44	2560	1312	2420
ML-DSA-65	4032	1952	3309
ML-DSA-87	4896	2592	4627

Tabla B.11: Tamaños (en bytes) de las claves y de las especificaciones de la firma ML-DSA

B.3. Firmas Digitales Basadas en Funciones Resumen

533. Existen dos algoritmos de firma digital basados en funciones hash aceptados en esta guía: XMSS y SLH-DSA. El algoritmo SLH-DSA (ver §B.3.1) es el resultado del proceso de estandarización del algoritmo SPHINCS$^+$, que superó la tercera ronda del NIST. Por otro lado, el algoritmo XMSS (ver §B.3.2) fue estandarizado con anterioridad a pesar de sus peculiaridades de uso para cubrir la necesidad de proveer una solución de firma con resistencia a la computación cuántica en escenarios donde es difícil realizar actualizaciones, como es el caso de las firmas de FW/SW mediante dispositivos hardware. En general, el problema computacional sobre el que se basan las firmas digitales basadas en funciones resumen se considera suficientemente robusto. Por este motivo su uso no requiere de hibridación con otras

firmas digitales utilizadas en la actualidad, a diferencia de lo que se recomienda con las firmas digitales basadas en retículos.

B.3.1. SLH-DSA

534. El algoritmo SPHINCS$^+$, candidato que superó la tercera ronda del NIST en 2022, fue seleccionado por esta institución como esquema de firma digital. Una vez que el NIST ha publicado el estándar de esta firma digital en [FIPS205], en esta sección comentaremos sus principales propiedades, así como los cambios realizados por el NIST. La nueva propuesta se denota como *Stateless Hash-Based Digital Signature Standard* o SLH-DSA.

535. SLH-DSA es un esquema de firma basado en hash sin estado que se construye utilizando como componentes otros esquemas de firma basados en hash: un esquema de firma de pocas veces, un bosque de subconjuntos aleatorios o FORS (*Forest Of Random Subsets*) y el esquema de firma XMSS (ver §B.3.2). XMSS se construye utilizando el esquema de firma única basado en hash WOTS$^+$ como componente. De hecho, los esquemas WOTS$^+$ y XMSS que se utilizan como componentes de SLH-DSA no son los mismos que los esquemas WOTS$^+$ y XMSS definidos en [RFC8391] y [SP800-208].

536. Conceptualmente, un par de claves SLH-DSA consta de un conjunto muy grande de pares de claves FORS, que para los conjuntos de parámetros de este estándar contiene 263, 264, 266 o 268 claves FORS, que se generan pseudoaleatoriamente a partir de una única semilla. El esquema de firma FORS permite que cada par de claves firme de forma segura una pequeña cantidad de mensajes (aproximadamente 10).

537. Por su parte, una firma SLH-DSA se crea calculando un hash aleatorio del mensaje, utilizando parte del resumen del mensaje resultante para seleccionar pseudoaleatoriamente una clave FORS y firmando la parte restante del resumen del mensaje con esa clave. Así, una firma SLH-DSA consta de la firma FORS junto con información que autentica la clave pública FORS. La información de autenticación se crea utilizando firmas XMSS.

538. Las principales diferencias entre SLH-DSA y SPHINCS$^+$ son las siguientes

    - Se han definido dos nuevos tipos de direcciones, WOTS_PRF y FORS_PRF, que se utilizan para la generación de valores de la clave secreta WOTS$^+$ y FORS.
    - Se ha añadido PK.seed como entrada a PRF para mitigar los ataques de múltiples claves.
    - Para los conjuntos de parámetros de las categorías 3 y 5 que usan SHA-2, la función SHA-256 fue reemplazada por SHA-512 en H_{msg}, PRF_{msg}, H y T_l,

dado que se descubrieron algunas debilidades al usar SHA-256 para obtener seguridad de categoría 5 (ver [Ste21], [Ant22] y [PKC22]).

- R y PK.seed se añadieron como entradas a MGF1 al calcular H_{msg} para los conjuntos de parámetros SHA-2 con el fin de paliar los ataques a la segunda preimagen de mensajes largos y múltiples objetivos.

539. En las secciones §3-§8 de [FIPS205] se presentan aspectos concretos y con mayor detalle de este estándar. En ellas se abordan aspectos generales del esquema de firma, funciones que se emplean en su implementación, la firma única Winternitz Plus, la firma XMSS, el hiperárbol SLH-DSA y el bosque de subconjuntos aleatorios. También se incluyen funciones que son necesarias para la implementación de los diferentes algoritmos. No obstante, a continuación se comentan de forma genérica algunas de estas características.

540. Este se diferencia del primero en el hecho de que incluye un paso de prehash adicional antes de la firma. Al igual que el ML-DSA, HashML-DSA también consta de tres algoritmos: el de generación de claves, que es el mismo algoritmo utilizado para ML-DSA, ML-DSA.KeyGen (Algoritmo 20), el de firma, HashML-DSA.Sign (Algoritmo 25), y el de verificación, HashML-DSA.Verify (Algoritmo 26).

541. Como ya hemos dicho, SLH-DSA utiliza el hiperárbol y las claves FORS para crear un esquema de firma basado en hash sin estado. La clave privada SLH-DSA contiene un valor inicial secreto y una clave PRF secreta. La clave pública está formada por un identificador de clave, PK.seed, y la raíz del hiperárbol. Una firma SLH-DSA se crea aplicando un hash al mensaje, utilizando h bits del resumen del mensaje para seleccionar una clave FORS, $h - h'$ bits para seleccionar un árbol XMSS en la capa más baja y h' bits para seleccionar una clave WOTS$^+$ (y la clave FORS correspondiente) de ese árbol. Además, se firman ka bits del resumen del mensaje con la clave FORS. Aunque sólo se utilizan bits $h + ka$ bits del resumen del mensaje, la implementación se simplifica extrayendo los bits necesarios de un resumen ligeramente más grande.

542. La clave pública de SLH-DSA contiene dos elementos: el primero es una semilla pública, PK.seed, de n bytes, que se utiliza en muchas llamadas a funciones hash para proporcionar separación de dominio entre diferentes pares de claves SLH-DSA. El segundo es la clave pública del hiperárbol, es decir, la raíz del árbol XMSS de la capa superior, PK.seed, que se genera utilizando un generador de bits aleatorios aprobado que admita, al menos, $8n$ bits de seguridad.

543. Por su parte, la clave privada SLH-DSA contiene dos valores secretos aleatorios de n bytes: uno de ellos, SK.seed, se utiliza para generar todos los elementos de clave privada de WOTS$^+$ y FORS; mientras que el otro, SK.prf, se emplea para generar un valor aleatorio para el hash aleatorio del mensaje en SLH-DSA. La clave privada incluye, además, una copia de la clave pública. Tanto SK.seed como SK.prf

se generan utilizando un RBG aprobado de modo que admita, al menos, $8n$ bits de seguridad.

544. De forma similar a como ya se ha comentado para los estándares ML-KEM y ML-DSA, también para SLH-DSA se consideran unos algoritmos internos y otros externos, que vienen a cumplir la misma labor ya mencionada.

545. A continuación se describen los algoritmos (funciones) internos y externos de generación de claves, generación de firmas y verificación de firmas de SLH-DSA. En las funciones en las que se requiere aleatoriedad, los valores aleatorios se proporcionan como entradas para las funciones. Las interfaces especificadas se utilizarán cuando se realicen pruebas de implementaciones de SLH-DSA a través del programa de validación de algoritmos criptográficos. La función de generación de claves interna también se puede utilizar para obtener la garantía de posesión de la clave privada a través de la regeneración.

546. Las claves públicas SLH-DSA contienen dos elementos: una semilla pública PK.seed de n bytes, que se utiliza en muchas llamadas a funciones hash para proporcionar separación de dominios entre diferentes pares de claves SLH-DSA, y la clave pública del hiperárbol (es decir, la raíz del árbol XMSS de la capa superior), PK.seed, que se generará utilizando un generador de bits aleatorios aprobado donde la instanciación del generador de bits aleatorios admita al menos $8n$ bits de seguridad. La clave privada SLH-DSA contiene dos valores secretos aleatorios de n bytes: el primero, SK.seed, se utiliza para generar todos los elementos de clave privada de WOTS$^+$ y FORS. El segundo, SK.prf, se utiliza para generar un valor de aleatorización para el hash aleatorio del mensaje en SLH-DSA. La clave privada también incluye una copia de la clave pública. Tanto SK.seed como SK.prf se generarán utilizando un generador de bits aleatorios aprobado, donde la instanciación del generador de bits aleatorios admita al menos $8n$ bits de seguridad.

547. La firma SLH-DSA tiene dos variantes: protegida y determinista, cuyas claves solo deben usarse para la generación y verificación de firmas digitales SLH-DSA protegidas y deterministas, respectivamente.

548. El Algoritmo 27 presenta la función interna para la generación de las claves.

Algoritmo 27 Generación de claves interno: slh_keygen_internal(SK.seed, SK.prf,PK.seed)

Genera un par de claves para SLH-DSA.
Entrada: semilla secreta SK.seed, clave PRF, SK.prf, y semilla pública, PK.seed.
Salida: par de claves, (SK, PK), para SLH-DSA.

1. ADRS $\leftarrow$ toByte$(0, 32)$
2. ADRS.setLayerAddress$(d-1)$
3. PK.root $\leftarrow$ xmss_node(SK.seed, $0, h'$, PK.seed, ADRS)
4. **devuelve** ((SK.seed, SK.prf, PK.seed, PK.root), (PK.seed, PK.root))

549. por su parte, el Algoritmo 28 muestra el procedimiento para generar las claves para SLH-DSA. Como se puede apreciar, en las líneas 1–3 se generan los valores aleatorios para las claves privada y pública; mientras que la línea 7 llama al algoritmo slh_keygen_internal para calcular PK.root y devolver la clave privada y pública. Además, PK.seed, SK.seed y SK.prf se generarán utilizando un generador de bits aleatorios aprobado, donde la instanciación del generador de bits aleatorios admita al menos $8n$ bits de seguridad.

Algoritmo 28 Generación de claves: slh_keygen()

Genera un par de claves para SLH-DSA. *Salida:* par de claves, (SK, PK), para SLH-DSA.

1. SK.seed $\xleftarrow{\$} \mathbb{B}^n$
2. SK.prf $\xleftarrow{\$} \mathbb{B}^n$
3. PK.seed $\xleftarrow{\$} \mathbb{B}^n$
4. **if** SK.seed $==$ NULL**or** SK.prf $==$ NULL**or** PK.seed $==$ NULL **then**
5. **devuelve** $\perp$
6. **end if**
7. **devuelve** slh_keygen_internal$(SK.seed, SK.prf, PK.seed)$

550. Por otra parte, en [FIPS205] se define, al igual que en el caso de las firmas ML-DSA, además del esquema puro de generación de una firma, un esquema de firma conocido como prehash y denotado por hashslh-dsa. Ambas versiones utilizan los correspondientes algoritmos internos para firmar y para verificar las firmas (los de

generación de claves son los mismos para ambas versiones); si bien difieren en cómo se crean las entradas a tales algoritmos internos. En lo que sigue, se presentarán, en primer lugar, los algoritmos internos y los conocidos como puros para la generación y verificación de las firmas y, más tarde, se mostrarán los correspondientes algoritmos para hashslh-dsa.

551. Una firma SLH-DSA consta de una cadena aleatoria de n bits, R, una firma FORS de $k(1+a)n$ bytes, SIG_{FORS}, y una firma de hiperárbol de $(h+d\cdot len)n$ bytes, SIG_{HT}.

552. El procedimiento para elaborar una de tales firmas se muestra como Algoritmo 30, que crea un resumen de mensaje de m bytes (líneas 2–5). De hecho, se utiliza un PRF para crear un generador aleatorio de mensajes (línea 3) y se aplica un hash junto con el mensaje para crear el resumen (línea 5). Posteriormente, se extraen los bits del resumen del mensaje para firmarlos con la clave FORS (línea 6), para seleccionar un árbol XMSS (líneas 7 y 9) y para seleccionar una clave WOTS^+ y la clave FORS correspondiente dentro de ese árbol XMSS (líneas 8 y 10). A continuación, se calcula la firma FORS (líneas 11–14) y se obtiene la clave pública FORS correspondiente (línea 16). Finalmente, se firma la clave pública FORS (línea 17).

553. El generador aleatorio de mensajes se puede configurar de forma determinista o no determinista, dependiendo de si se proporciona $addrnd$ como entrada. Para la variante protegida, $addrnd$ se proporciona como entrada y opt_rnd se establece en $addrnd$. La variante protegida es la predeterminada y se debe utilizar en plataformas donde los ataques de canal lateral sean una preocupación. Cuando se utiliza la versión protegida, $addrnd$ debe ser un valor aleatorio de n bytes. Lo ideal es que $addrnd$ se genere mediante un generador de bits aleatorios aprobado, aunque se pueden utilizar otros métodos para generar nuevos valores aleatorios. Por su parte, para la variante determinista, $addrnd$ no se proporciona como entrada y opt_rnd se establece en PK.seed, lo que hace que la firma sea determinista. La variante determinista está disponible para plataformas donde no hay un generador de bits aleatorios disponible.

Algoritmo 29 Elaboración de la firma digital interna: slh_sign_internal(M, SK)

Genera una firma SLH-DSA.
Entrada: mensaje M, la clave privada SK = (SK.seed, SK.prf, PK.seed, PK.root) y aleatoriedad opcional $addrnd$.
Salida: la firma SIG.

1. ADRS $\leftarrow$ toByte$(0, 32)$
2. $opt_rand \leftarrow addrand$
3. $R \leftarrow \text{PRF}_{msg}(\text{SK.prf}, opt_rand, M)$
4. SIG $\leftarrow R$
5. $digest \leftarrow \text{H}_{msg}(R, \text{PK.seed}, \text{PK.root}, M)$
6. $md \leftarrow digest\left[0 : \left\lceil \frac{ka}{8} \right\rceil\right]$
7. $tmp_idx_{tree} \leftarrow digest\left[\left\lceil \frac{ka}{8} \right\rceil : \left\lceil \frac{ka}{8} \right\rceil + \left\lceil \frac{h-h/d}{8} \right\rceil\right]$
8. $tmp_idx_{leaf} \leftarrow digest\left[\left\lceil \frac{ka}{8} \right\rceil + \left\lceil \frac{h-h/d}{8} \right\rceil : \left\lceil \frac{ka}{8} \right\rceil + \left\lceil \frac{h-h/d}{8} \right\rceil + \left\lceil \frac{h}{8d} \right\rceil\right]$
9. $idx_{tree} \leftarrow \text{toInt}\left(tmp_idx_{tree}, \left\lceil \frac{h-h/d}{8} \right\rceil\right) \pmod{2^{h-h/d}}$
10. $idx_{leaf} \leftarrow \text{toInt}\left(tmp_idx_{leaf}, \left\lceil \frac{h}{8d} \right\rceil\right) \pmod{2^{h/d}}$
11. ADRS.setTreeAddress(idx_{tree})
12. ADRS.setTypeAndClear(FORS_TREE)
13. ADRS.setKeyPairAddress (idx_{leaf})
14. $\text{SIG}_{FORS} \leftarrow$ fors_sign$(md, \text{SK.seed}, \text{PK.seed}, \text{ADRS})$
15. SIG $\leftarrow$ SIG$||\text{SIG}_{FORS}$
16. $\text{PK}_{FORS} \leftarrow$ fors_pkFromSig $(\text{SIG}_{FORS}, md, \text{PK.seed}, \text{ADRS})$
17. $\text{SIG}_{HT} \leftarrow$ ht_sign $(\text{PK}_{FORS}, \text{SK.seed}, \text{PK.seed}, idx_{tree}, idx_{leaf})$
18. SIG $\leftarrow$ SIG$||\text{SIG}_{HT}$
19. **devuelve** SIG

554. Las versiones de firma pura y prehash utilizan slh_sign_internal, pero difieren en cómo se crea la entrada de mensaje a slh_sign_internal a partir del contenido que se va a firmar. En la versión pura, el contenido se firma con slh_sign_internal junto con cierta información de separación de dominios. En la versión prehash, un hash del contenido se firma con slh_sign_internal junto con cierta información de separación de dominios. Ambas versiones toman el contenido que se va a firmar, la clave privada y un contexto como entrada. La versión prehash, como luego se verá, también toma como entrada una función hash o XOF que se va a utilizar para hacer un prehash del contenido que se va a firmar. La cadena de contexto tiene una longitud máxima de 255 bytes. De forma predeterminada, el contexto es la cadena vacía. Sin embargo, las aplicaciones pueden especificar el uso de una cadena de contexto no vacía.

Algoritmo 30 Elaboración de la firma digital interna: slh_sign_internal(M, ctx, SK)

Genera una firma SLH-DSA pura.
Entrada: mensaje M, cadena de contexto, ctx, y clave privada SK.
Salida: firma SIG.

1. **if** $|ctx| > 255$**then**
2. **devuelve** $\perp$
3. **end if**
4. $addrnd \xleftarrow{\$} \mathbb{B}^n$
5. **if** $addrnd =$ NULL**then**
6. **devuelve** $\perp$
7. **end if**
8. $M' \leftarrow$ toByte$(0,1)||$toByte$(|ctx|,1)||ctx||M$
9. SIG $\leftarrow$ slh_sign_internal$(M', \text{SK}, addrnd)$
10. **devuelve** SIG

555. Al igual que con la elaboración de la firma, la verificación interna con SLH-DSA (ver Algoritmo 31) calcula un resumen del mensaje (línea 8) y luego extrae md (línea 9), idx_{tree} (líneas 10 y 12) e idx_{leaf} (líneas 11 y 13) del resumen. A continuación se calcula una clave pública FORS candidata (línea 17) y se verifica la firma con la clave FORS (línea 18). Si esta verificación de firma es correcta, entonces la clave pública FORS también lo era y la firma SIG del mensaje M es válida.

Algoritmo 31 Verificación de la firma digital interna: slh_verify_internal$(M, \text{SIG}, \text{PK})$

Verifica una firma SLH-DSA.
Entrada: mensaje M, firma SIG y la clave pública, PK = (PK.seed, PK.root).
Salida: valor booleano.

1. **if** $|\text{SIG}| \neq (1 + k(1 + a) + h + d \cdot len)n$ **then**
2. **devuelve** falso
3. **end if**
4. ADRS $\leftarrow$ toByte$(0, 32)$
5. $R \leftarrow$ SIG.getR()
6. $\text{SIG}_{FORS} \leftarrow$ SIG.getSIG_FORS()
7. $\text{SIG}_{HT} \leftarrow$ SIG.getSIG_HT()
8. $digest \leftarrow \text{H}_{msg}(R, \text{PK.seed}, \text{PK.root}, M)$
9. $md \leftarrow digest\,[0 : \lceil \frac{ka}{8} \rceil]$
10. $tmp_idx_{tree} \leftarrow digest\left[\lceil \frac{ka}{8} \rceil : \lceil \frac{ka}{8} \rceil + \left\lceil \frac{h-h/d}{8} \right\rceil\right]$
11. $tmp_idx_{leaf} \leftarrow digest\left[\lceil \frac{ka}{8} \rceil + \left\lceil \frac{h-h/d}{8} \right\rceil : \lceil \frac{ka}{8} \rceil + \left\lceil \frac{h-h/d}{8} \right\rceil + + \lceil \frac{h}{8d} \rceil\right]$
12. $idx_{tree} \leftarrow \text{toInt}\left(tmp_idx_{tree}, \left\lceil \frac{h-h/d}{8} \right\rceil\right) \pmod{2^{h-h/d}}$
13. $idx_{leaf} \leftarrow \text{toInt}\left(tmp_idx_{leaf}, \lceil \frac{h}{8d} \rceil\right) \pmod{2^{h/d}}$
14. ADRS.setTreeAddress (idx_{tree})
15. ADRS.setTypeAndClear(FORS_TREE)
16. ADRS.setKeyPairAddress (idx_{leaf})
17. $\text{PK}_{FORS} \leftarrow$ fors_pkFromSig $(\text{SIG}_{FORS}, md, \text{PK.seed}, \text{ADRS})$
18. **devuelve** ht_verify $(\text{PK}_{FORS}, \text{SIG}_{HT}, \text{PK.seed}, idx_{tree}, idx_{leaf}, \text{PK.root})$

556. Al igual que antes, las versiones de verificación de firma pura y prehash utilizan slh_verify_internal, pero difieren en cómo se crea la entrada de mensaje al mismo.

557. El algoritmo de verificación puro con SLH-DSA se muestra a continuación como Algoritmo 32.

Algoritmo 32 Verificación de la firma digital: slh_verify$(M, \text{SIG}, ctx, \text{PK})$

Verifica una firma SLH-DSA pura.
Entrada: mensaje M, firma SIG, cadena de contexto, ctx, y clave pública, PK.
Salida: valor booleano.

1. **if** $|ctx| > 255$ **then**
2. **devuelve** falso
3. **end if**
4. $M' \leftarrow \text{toByte}(0, 1)||\text{toByte}(|ctx|, 1)||ctx||M$
5. **devuelve** slh_verify_internal$(M', \text{SIG}, \text{PK})$

558. Los algoritmos de elaboración de firma y verificación de la misma para el caso HashSLH-DSA se presentan a continuación.

559. En la versión prehash de la firma, la entrada de mensaje a slh_sign_internal es el resultado de aplicar una función hash o un XOF al contenido que se va a firmar. La salida de la función hash o XOF se antepone a un separador de dominio de un byte (un byte que indica la longitud de la cadena de contexto), la cadena de contexto y la codificación de las reglas de codificación distinguidas de la función hash o el identificador de objeto (OID) del XOF. El separador de dominio tiene un valor de uno para la firma prehash. La codificación del OID incluye la etiqueta y la longitud. El Algoritmo 33 muestra las codificaciones de los OID para SHA-256, SHA-512, SHAKE128 y SHAKE256. Sin embargo, hash_slh_sign se puede utilizar con otras funciones hash o XOF. Debe recordarse que SHA-256 y SHAKE128 solo son apropiados para ser utilizados con conjuntos de parámetros SLH-DSA que están en la categoría de seguridad 1.

Algoritmo 33 Elaboración de la firma digital prehash: hash_slh_sign$(M, ctx, \text{PH}, \text{SK})$

Genera una firma SLH-DSA prehash.
Entrada: mensaje, M, cadena de contexto, ctx, función prehash, PH, y clave privada SK.
Salida: firma SIG para SLH-DSA.

1. **if** $|ctr| > 255$ **then**
2. **devuelve** $\perp$
3. **end if**
4. $addrnd \xleftarrow{\$} \mathbb{B}^n$
5. **if** $addrnd ==$ NULL **then**
6. **devuelve** $\perp$
7. **end if**
8. **switch** PH **do**
9. **case** SHA-256
10. OID $\leftarrow$ toByte$(0x0609608648016503040201, 11)$
11. $\text{PH}_M \leftarrow$ SHA-256(M)
12. **case** SHA-512
13. OID $\leftarrow$ toByte$(0x0609608648016503040203, 11)$
14. $\text{PH}_M \leftarrow$ SHA-512(M)
15. **case** SHAKE128
16. OID $\leftarrow$ toByte$(0x060960864801650304020B, 11)$
17. $\text{PH}_M \leftarrow$ SHAKE128$(M, 256)$
18. **case** SHAKE256
19. OID $\leftarrow$ toByte$(0x060960864801650304020C, 11)$
20. $\text{PH}_M \leftarrow$ SHAKE128$(M, 512)$
21. **end switch**
22. **case** ...
23. ...
24. **end switch**
25. $M' \leftarrow$ toByte$(1, 1)||$toByte$(|ctx|, 1)||ctx||$OID$||\text{PH}_M)$
26. SIG $\leftarrow$ slh_sign_internal$(M', \text{SK}, addrnd)$
27. **devuelve** SIG

560. El Algoritmo 34 presenta el proceso a seguir para la verificación de una firma prehash.

Algoritmo 34 Verificación de la firma digital prehash: hash_slh_verify$(M, \text{SIG}, ctx, \text{PH}, \text{PK})$

Verifica una firma SLH-DSA prehash.
Entrada: mensaje, M, firma, SIG, cadena de contexto, ctx, función prehash, PH, y clave pública, PK.
Salida: valor booleano.

1. **if** $|ctr| > 255$ **then**
2. **devuelve** falso
3. **end if**
4. **switch** PH **do**
5. **case** SHA-256
6. OID $\leftarrow$ toByte$(0x0609608648016503040201, 11)$
7. $\text{PH}_M \leftarrow$ SHA-256(M)
8. **case** SHA-512
9. OID $\leftarrow$ toByte$(0x0609608648016503040203, 11)$
10. $\text{PH}_M \leftarrow$ SHA-512(M)
11. **case** SHAKE128
12. OID $\leftarrow$ toByte$(0x060960864801650304020B, 11)$
13. $\text{PH}_M \leftarrow$ SHAKE128$(M, 256)$
14. **case** SHAKE256
15. OID $\leftarrow$ toByte$(0x060960864801650304020C, 11)$
16. $\text{PH}_M \leftarrow$ SHAKE128$(M, 512)$
17. **end switch**
18. **case** ...
19. ...
20. **end switch**
21. $M' \leftarrow$ toByte$(1, 1)||$toByte$(|ctx|, 1)||ctx||\text{OID}||\text{PH}_M)$
22. **devuelve** slh_verify_internal$(M', \text{SIG}, \text{PK})$

561. La versión publicada por el NIST solo aprueba el uso de 12 de los 36 conjuntos de parámetros definidos en [HBD+20], dado que solo se aceptan los casos simples en los que las funciones criptográficas empleadas sean SHA-2 o SHAKE.

562. Un conjunto de parámetros para SLH-DSA son los parámetros para WOTS$^+$ (n y $\lg_w$), el hiperárbol para XMSS y SLH-DSA (h y d) y FORS (k y a), a la vez que los valores para las funciones H_{msg}, PRF, PRF_{msg}, F, H, and T_l. SLH-DSA utiliza un parámetro adicional m, que es la longitud en bytes del resumen del mensaje y

que se determina como sigue:

$$m = \left\lceil \frac{h - h'}{8} \right\rceil + \left\lceil \frac{h'}{8} \right\rceil + \left\lceil \frac{ka}{8} \right\rceil$$

563. La Tabla B.12 presenta los conjuntos de parámetros cuyo uso está aprobado. Cada uno de los nombres, se señala la familia de funciones hash (SHA2 o SHAKE) que se utiliza, la longitud en bits del parámetro de seguridad, n, y si el conjunto de parámetros fue diseñado para crear firmas relativamente pequeñas (s) o generar firmas relativamente rápidas (f). Además, Cat. hace referencia a la categoría de seguridad establecido, PK es el tamaño, en bytes, de la clave pública y SIG es el tamaño, también en bytes de la firma.

Nombre	n	h	d	h'	a	k	$\lg_w$	m	Cat.	PK	SIG
SLH-DSA-SHA2-128s SLH-DSA-SHAKE-128s	16	63	7	9	12	14	4	30	1	32	7856
SLH-DSA-SHA2-128f SLH-DSA-SHAKE-128f	16	66	22	3	6	33	4	34	1	32	17088
SLH-DSA-SHA2-192s SLH-DSA-SHAKE-192s	24	63	7	9	14	17	4	39	3	48	16224
SLH-DSA-SHA2-192f SLH-DSA-SHKE-192f	24	66	22	3	8	33	4	42	3	48	35664
SLH-DSA-SHA2-256s SLH-DSA-SHAKE-256s	32	64	8	8	14	22	4	47	5	64	29792
SLH-DSA-SHA2-256f SLH-DSA-SHAKE-256f	32	68	17	4	9	35	4	49	5	64	49856

Tabla B.12: Conjuntos de parámetros para SLH-DSA

564. Debe tenerse en cuenta, como ya se ha mencionado con antelación, que los conjuntos de parámetros incluidos en la Tabla B.12 se diseñaron para cumplir con las categorías de fortaleza de seguridad definidas por NIST en su convocatoria de propuestas original con respecto a la imposibilidad de falsificación existencial bajo ataque de mensaje elegido (EUF-CMA) cuando cada par de claves se usa para firmar, como máximo, 2^{64} mensajes.

565. Este enfoque significa que la fortaleza de la seguridad no se describe como es tradicional, en el sentido de afirmar que tiene, por ejemplo, "128 bits de seguridad". De hecho, lo que se afirma es que cada conjunto de parámetros es, al menos, tan seguro como un cifrador en bloque con un tamaño de clave determinado, esto es, que los recursos computacionales necesarios para vulnerar SLH-DSA son mayores o iguales a los recursos computacionales necesarios para descifrar el cifrado de bloque. En particular, los conjuntos de parámetros con $n = 16$ ofrecen una seguridad de categoría 1, con $n = 24$ son de categoría 3 y con $n = 32$ son de categoría 5.

566. Como se puede ver, hay seis conjuntos de valores para n, $\lg_w$, h, d, k y a cuyo uso está aprobado. Para los conjuntos de parámetros con SHAKE o SHA2, se

establecen las instancias de las funciones que se especifican a continuación. Así, para los conjuntos de parámetros con SHAKE, se consideran las instancias para las siguientes funciones:

- $\mathsf{H}_{msg}(R, \mathsf{PK.seed}, \mathsf{PK.root}, M) = \mathsf{SHAKE256}(R||\mathsf{PK.seed}||\mathsf{PK.root}||M, 8m)$
- $\mathsf{PRF}(\mathsf{PK.seed}, \mathsf{SK.seed}, \mathsf{ADRS}) = \mathsf{SHAKE256}(\mathsf{PK.seed}||\mathsf{ADRS}||\mathsf{SK.seed}, 8n)$
- $\mathsf{PRF}_{msg}(\mathsf{SK.prf}, opt_rand, M) = \mathsf{SHAKE256}(\mathsf{SK.prf}||opt_rand||M, 8n)$
- $\mathsf{F}(\mathsf{PK.seed}, \mathsf{ADRS}, M_1) = \mathsf{SHAKE256}(\mathsf{PK.seed}||\mathsf{ADRS}||M_1, 8n)$
- $\mathsf{H}(\mathsf{PK.seed}, \mathsf{ADRS}, M_2) = \mathsf{SHAKE256}(\mathsf{PK.seed}||\mathsf{ADRS}||M_2, 8n)$
- $\mathsf{T}_l(\mathsf{PK.seed}, \mathsf{ADRS}, M_l) = \mathsf{SHAKE256}(\mathsf{PK.seed}||\mathsf{ADRS}||M_l, 8n)$

567. Para los conjuntos de parámetros SHA2, se establecen las instancias de las funciones siguientes si $n = 16$ (categoría 1):

- $\mathsf{H}_{msg}(R, \mathsf{PK.seed}, \mathsf{PK.root}, M) =$
  $\mathsf{MGF1\text{-}SHA\text{-}256}(R||\mathsf{PK.seed}||\mathsf{SHA\text{-}256}(R||\mathsf{PK.seed}||\mathsf{PK.root}||M), m)$
- $\mathsf{PRF}(\mathsf{PK.seed}, \mathsf{SK.seed}, \mathsf{ADRS}) =$
  $\mathsf{Trunc}_n(\mathsf{SHA\text{-}256}(\mathsf{PK.seed}||\mathsf{toByte}(0, 64 - n)||\mathsf{ADRS}||\mathsf{SK.seed}))$
- $\mathsf{PRF}_{msg}(\mathsf{SK.prf}, opt_rand, M) =$
  $\mathsf{Trunc}_n(\mathsf{HMAC\text{-}SHA\text{-}256}(\mathsf{SK.prf}, opt_rand||M))$
- $\mathsf{F}(\mathsf{PK.seed}, \mathsf{ADRS}, M_1) =$
  $\mathsf{Trunc}_n(\mathsf{SHA\text{-}256}(\mathsf{PK.seed}||\mathsf{toByte}(0, 64 - n)||\mathsf{ADRS}^c||M_1))$
- $\mathsf{H}(\mathsf{PK.seed}, \mathsf{ADRS}, M_2) =$
  $\mathsf{Trunc}_n(\mathsf{SHA\text{-}256}(\mathsf{PK.seed}||\mathsf{toByte}(0, 64 - n)||\mathsf{ADRS}^c||M_2))$
- $\mathsf{T}_l(\mathsf{PK.seed}, \mathsf{ADRS}, M_l) =$
  $\mathsf{Trunc}_n(\mathsf{SHA\text{-}256}(\mathsf{PK.seed}||\mathsf{toByte}(0, 64 - n)||\mathsf{ADRS}^c||M_l)$

568. Finalmente, si $n = 24$ o $n = 32$ (categorías 3 y 5), se consideran las instancias de las siguientes funciones:

- $Hmsg(R, \mathsf{PK.seed}, \mathsf{PK.root}, M) =$
  $\mathsf{MGF1\text{-}SHA\text{-}512}(R||\mathsf{PK.seed}||\mathsf{SHA\text{-}512}(R||\mathsf{PK.seed}||\mathsf{PK.root}||M), m)$
- $PRF(\mathsf{PK.seed}, \mathsf{SK.seed}, \mathsf{ADRS}) =$
  $\mathsf{Trunc}_n(\mathsf{SHA\text{-}256}(\mathsf{PK.seed}||\mathsf{toByte}(0, 64 - n)||\mathsf{ADRS}^c||\mathsf{SK.seed}))$

- $\text{PRF}_{msg}(\text{SK.prf}, opt_rand, M) =$
  $\text{Trunc}_n(\text{HMAC-SHA-512}(\text{SK.prf}, opt_rand||M))$
- $\text{F}(\text{PK.seed}, \text{ADRS}, M_1) =$
  $\text{Trunc}_n(\text{SHA-256}(\text{PK.seed}||\text{toByte}(0, 64-n)||\text{ADRS}^c||M_1))$
- $\text{H}(\text{PK.seed}, \text{ADRS}, M_2) =$
  $\text{Trunc}_n(\text{SHA-512}(\text{PK.seed}||\text{toByte}(0, 128-n)||\text{ADRS}^c||M_2))$
- $T_l(\text{PK.seed}, \text{ADRS}, M_l) =$
  $\text{Trunc}_n(\text{SHA-512}(\text{PK.seed}||\text{toByte}(0, 128-n)||\text{ADRS}^c||M_l))$

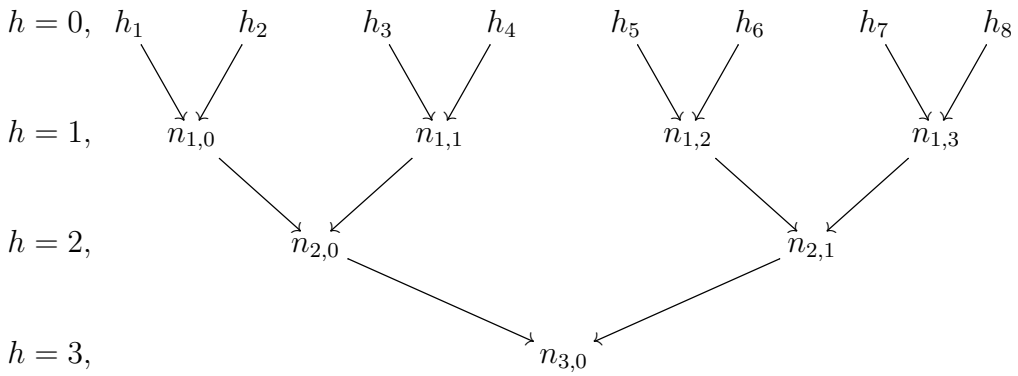
B.3.2. XMSS

569. El algoritmo de firma XMSS fue propuesto, como ya se ha dicho, por Buchmann et al. [BDH11] y desarrollado posteriormente en [RFC8391] y [SP800-208] como un algoritmo extendido del esquema de firma de Merkle (MSS) [Mer89]. Ambos esquemas se conocen como esquemas de firmas basadas en *hashes* o HBS (*Hash-Based Signatures*). Las HBS son unos de los primeros protocolos criptográficos asimétricos propuestos y se basan en el esquema de firma única de Lamport [Lam79]. Su fundamento son los conocidos árboles de Merkle, que a su vez son un tipo particular de árbol binario.

570. Para comprender el esquema XMSS, presentaremos brevemente el esquema MSS (basado en el esquema de Lamport), que se puede utilizar para firmar un número limitado de mensajes, de hecho, una potencia de 2 ($N = 2^H$).

571. Un árbol binario es una estructura en forma de árbol de modo que cada uno del mismo tiene exactamente tres aristas, dos de las cuales van al nivel más cercano a las hojas y la otra va a la raíz. Las hojas tienen una única arista y la raíz tiene solo dos aristas.

572. Un ejemplo sencillo de árbol binario con $2^H = 2^3 = 8$ hojas se muestra a continuación:



En general, los nodos se denotan como $n_{h,k}$, donde h es el número del piso y k es el orden dentro de un piso, de izquierda a derecha, de modo que se tiene $0 \leq k \leq 2^{H-h} - 1$ para el piso número h.

573. Para implementar una HBS se pueden usar árboles de Merkle junto con un esquema de firma única u OTS (*One Time Signature*), como la de Lamport. Un OTS es un esquema de firma con una clave privada que se usa para firmar un mensaje y la clave pública correspondiente se usa para verificar dicha firma. El problema es que la clave privada solo se puede usar una vez para firmar un mensaje.

574. La propuesta de Lamport se puede resumir de la siguiente manera [Lam79]: dado un mensaje $m \in \{0,1\}^n$ y una función unidireccional $F\colon \{0,1\}^k \to \{0,1\}^k$, el firmante genera una clave privada y la clave pública correspondiente. La clave privada consta de $2\,n$ valores aleatorios de k bits cada uno, $s_{i,j}$ para $i \in \{1,\ldots,n\}$ y $j \in \{0,1\}$. La clave es el resultado de aplicar la función unidireccional F a cada valor secreto de modo que se tiene $p_{i,j} = F(s_{i,j})$ y la clave pública es $pk = (p_{1,0}, p_{1,1}, \ldots, p_{n,0}, p_{n,1})$. Dado un mensaje, m, el firmante revela selectivamente los secretos s_{i,m_i}, siendo m_i es el i-ésimo bit de m. Esta lista de valores componen la firma buscada. Finalmente, dada firma, el verificador debe aplicar F a cada secreto revelado s_{i,m_i} de modo que la firma se considera válida si $F(s_{i,m_i}) = p_{i,m_i}$ para todos los bits de m.

575. Un árbol HBS es un árbol Merkle cuyas hojas son las claves públicas del OTS, y cada nodo interno del árbol consiste en el *hash* de sus dos hijos. La raíz del árbol es la clave pública de la construcción de Merkle.

576. Dados 2^H pares de claves privadas/públicas, (s_i, p_i), con $0 \leq i \leq 2^H - 1$, si $\mathfrak{h}$ es una función *hash*, en el árbol de Merkle se tendría que

$$n_{0,k} = \mathfrak{h}(P_k),\ 0 \leq k \leq 2^H - 1,$$
$$n_{h,k} = \mathfrak{h}\left(n_{h-1,2k} \mid\mid n_{h-1,2k+1}\right),$$

para $1 \leq h \leq H$, $0 \leq k \leq 2^{H-h} - 1$, siendo $||$ la concatenación. Una vez completado el árbol, el usuario publica como clave pública global el valor del nodo raíz, $n_{H,0}$, y sirve como compromiso de todo el conjunto de claves públicas, p_i.

577. De forma más precisa, el esquema de Merkle utiliza 2^H instancias de OTS, cada una con un par de claves públicas y privadas, (p_i, s_i), y construye un árbol de Merkle cuyas hojas son los *hashes* de las claves públicas de las instancias de OTS, y cada nodo es el *hash* de sus dos nodos secundarios. La clave pública consiste en la raíz del árbol de Merkle y la clave privada contiene las claves privadas de todas las instancias de OTS de 2^H. La firma de un mensaje m consta de un índice i que especifica una instancia OTS (p_i, s_i), la firma única σ_{OTS} en m bajo la clave p_i, la clave p_i y una ruta de autenticación A_i que se utiliza para verificar la validez de p_i.

578. La ventaja del esquema MSS es que se considera resistente a los algoritmos cuánticos, de hecho, solo depende de la existencia de funciones *hash* seguras.

579. Desde que Merkle anunció su propuesta, se ha publicado una gran cantidad de trabajos que intentan mejorar diferentes aspectos de MSS [DSS05, BCD+06, BDK+07, BDS08, BDS09], etc. En particular, es de destacar la propuesta XMSS de Buchmann [BDH11].

580. El esquema XMSS usa un OTS que solo puede firmar un mensaje con una clave, pero para superar esta limitación, se usa un árbol HBS que permite reducir la autenticidad de muchas claves de verificación OTS a una clave XMSS pública. Además, para minimizar los requisitos de almacenamiento, se utilizan generadores pseudo-aleatorios.

581. Para reducir el tamaño de la clave privada (que consiste en las claves privadas de las 2^H instancias OTS) se utiliza una función pseudo-aleatoria o PRF (*Pseudo-Random Function*), con una semilla maestra de n bits con el fin de generar una semilla OTS de n bits para cada instancia OTS, que a su vez se utiliza para generar la clave privada de esa instancia.

582. El OTS que usa XMSS es una variante de la OTS de Winternitz (W-OTS), llamada WOTS+ en [Mer89], [BDE+11] y [Hül13], que elimina el requisito de una función *hash* resistente a colisiones.

583. Además de los árboles binarios ya mencionados, también son de interés los árboles binarios no balanceados, llamados L-árboles [18]. Estos se utilizan exclusivamente para los hash de claves públicas WOTS+. Las ℓ hojas de un L-Tree son los elementos de una clave pública WOTS+ y el árbol se construye como los ya mencionados, con la salvedad de que el nodo izquierdo que no tiene un hermano derecho se eleva a un nivel superior del L-árbol hasta que se convierte en el hermano derecho de otro nodo. Los L-árboles tienen una altura de $\lceil \log \ell \rceil$ y, por lo tanto, necesitan $\lceil \log \ell \rceil$ máscaras.

584. Además, XMSS usa una estructura de L-árbol para reducir la clave pública OTS [BHH+15] y utiliza máscaras de cegamiento para las cadenas WOTS+ y para cada nodo en HBS y en el L-árbol. Las claves ciegas se generan de forma pseudo-aleatoria para cada nodo del árbol mientras se genera o verifica una firma.

585. Los parámetros públicamente conocidos para generar las claves de firma OTS son los siguientes:

- Los parámetros de seguridad son $n, w, m, p \in \mathbb{N}$, que corresponden, respectivamente, a la longitud del *hash* (en bytes), el parámetro Winternitz, la longitud del mensaje en bits y el número de cadenas Winternitz utilizadas en una sola operación OTS.
- Un familia de funciones $F(n) = \{f_K : \{0,1\}^n \to \{0,1\}^n | K \in \{0,1\}^n\}$.
- La altura del árbol, $H \in \mathbb{N}$ (recuérdese que XMSS permite hacer 2^H firmas usando un par de claves).

- Una función resumen $\mathfrak{h}_K$ elegida aleatoriamente con distribución uniforme de la familia $\mathfrak{H}(n) = \{\mathfrak{h}_K : \{0,1\}^{2n} \to \{0,1\}^n | K \in \{0,1\}^n\}$.
- Una cadena elegida al azar con la distribución uniforme, $x \in \{0,1\}^n$. La cadena x se usa para construir las claves de verificación única.

586. Para $K, x \in \{0,1\}^n$ y $e \in \mathbb{N}$, se define $f_K^e(x)$ como sigue:

$$f_K^0(x) = K, \quad f_K^{e-1}(x) = K' \text{ if } e > 0, \quad \text{y } f_K^e(x) = f_{K'}(x).$$

B.4. Recomendaciones para la Resistencia Cuántica

B.4.1. Transición segura de la precuántica a la postcuántica

587. Tal y como ya se ha señalado, en 1997, Shor publicó en [Sho97] dos algoritmos cuánticos capaces de vulnerar, en el momento de que se disponga de un ordenador cuántico con la suficiente capacidad de cómputo, los dos criptosistemas asimétricos más empleados en la actualidad: el RSA y los basados en curvas elípticas.

588. En [CCN22], el CCN presentó una colección de recomendaciones para una transición postcuántica segura, esto es, se detallaron las acciones que se deben llevar a cabo para protegerse, en la medida de lo posible, de esta amenaza cuántica.

589. Estas recomendaciones pueden resumirse de la siguiente manera:

- Plan de migración, que consta de los siguientes puntos:
  1. Determinar la información que debo proteger y hasta cuándo.
  2. Realizar un inventario exhaustivo de productos y cifradores que empleo para proteger mi información y mis activos.
  3. Analizar con rigor si tales productos y cifradores son o no resistentes a la computación cuántica.
  4. Establecer un plan de migración a las soluciones híbridas.
  5. Decidir qué nuevos productos necesito y cuánto tiempo requiero para su adquisición y despliegue.
  6. Determinar cuánto tiempo tengo disponible.
- Presentación de los algoritmos postcuánticos recomendados para ser tenidos en cuenta, tanto para el caso de los mecanismos de encapsulado de claves (PQC-KEM), de firmas digitales (PQC-DS), como de firmas basadas en hash para firmware:

1. CRYSTALS-Kyber (ahora ML-KEM).
   Su rendimiento es muy competitivo y su diseño es relativamente sencillo, por lo que es relevante para muchos casos de uso. Como se sabe, está basado en problemas definidos sobre retículos estructurados. El NIST ha publicado un estándar [FIPS203].
2. FrodoKEM.
   Este KEM es una variante más conservadora que Kyber y su diseño también es sencillo, dado que se basa en un problema de retículos no estructurados. Existe un proyecto de ISO para convertirlo en norma [ABD+23].
3. CRYSTALS-Dilithium (ahora ML-DSA).
   Al igual que Kyber, su rendimiento es muy competitivo y su diseño es relativamente sencillo. Está basado en problemas definidos sobre retículos estructurados. El NIST ha publicado un estándar [FIPS204].
4. Falcon.
   Posiblemente esta firma sea la que tenga el diseño más compacto y eficiente y también se basa en problemas sobre retículos estructurados. Es de destacar que su implementación necesita instrucciones particulares (coma flotante). El NIST publicará en breve un borrador de su estándar.
5. SPHINCS$^+$ (ahora SLH-DSA).
   Este esquema de firma es una variante sin estado de XMSS, por lo que su firma es más conservadora, esto es, sus hipóteis de seguridad son menores. Es menos competitivo en términos de rendimiento y compacidad. El NIST ha publicado un estándar [FIPS205].
6. XMSS.
   Como precursor de SPHINCS$^+$, es un esquema de firma con estado conservador y potencialmente tiene un número limitado de posibles firmas para cada de claves. Existe un estándar publicado en [RFC8391].

- Las longitudes de las claves, tanto para los algoritmos simétricos como para las funciones hash:

  1. AES-256.
  2. SHA2-384 y SHA2-512.
  3. SHA3-384 y SHA3-512.

- El uso de soluciones híbridas, es decir, combinar de modo simultáneo algoritmos postcuánticos con algoritmos precuánticos.
  Se trata de lograr las garantías criptográficas tradicionales que ofrece la criptografía actual, junto a las que ofrecen las soluciones resistentes a la computación cuántica.
  En particular, los acuerdos de clave deben usar al menos dos de los tres algoritmos siguientes: PreSharedKeys, EC-DH y PQC-KEM.

- Inclusión de las cuatro fases que muestran el calendario recomendado para proceder a una transición segura:

Fase 1 Empleo inmediato de las firmas basadas en hash para actualizaciones de firmware.

Fase 2 Uso de soluciones híbridas para proporcionar mayor defensa a la criptografía precuántica.

Fase 3 Empleo de soluciones híbridas con las garantías de seguridad proporcionadas por la criptografía postcuántica.

Fase 4 Adopción de los algoritmos criptográficos postcuánticos considerados resistentes a la computación cuántica.

590. Para cada uno de los algoritmos postcuánticos mencionados más arriba, se deben tener en cuenta las siguientes consideraciones:

- CRYSTALS-Kyber (ahora ML-KEM).

1. Para Kyber no se deben modificar los parámetros de la instancia estandarizada a menos que esté justificado.
2. Utilizar el nivel de seguridad más alto posible, preferiblemente el nivel 5 (es decir, equivalente a AES- 256), aunque es también aceptable el nivel 3 (equivalente a AES-192).
3. Utilizar claves efímeras siempre que sea posible, dado que previene muchos ataques, como los de fallo de descifrado.
4. Utilizar la versión semánticamente segura (IND-CCA) estandarizada por NIST.
5. Hay determinados casos, como en protocolos autenticados demostrables, en los que la versión IND-CPA en modo estático aún puede ser segura. Sin embargo, en estos casos, no debe estar disponible ningún oráculo de descifrado (ni siquiera en el canal lateral).

- FrodoKEM.
- CRYSTALS-Dilithium (ahora ML-DSA) y Falcon.

1. Para las PQC-DS no se deben modificar los parámetros de la instancia estandarizada a menos que esté justificado.
2. Utilizar el nivel de seguridad más alto posible, preferiblemente el nivel 5 (es decir, equivalente a AES-256).

3. Prestar atención y centrarse en el diseño establecido para evitar ataques de mal uso. Las distribuciones gaussianas en Falcon juegan un papel importante en la seguridad y no deben reemplazarse.
4. Para Falcon, las contramedidas de canal lateral son difíciles de aplicar y las investigaciones han demostrado que los ataques de canal lateral pueden vulnerar implementaciones desprotegidas de Falcon.

- SPHINCS$^+$ (ahora SLH-DSA) y XMSS.

1. No modificar los parámetros de la instancia estandarizada a menos que esté justificado.
2. Utilizar el nivel de seguridad más alto posible, preferiblemente el nivel 5 (es decir, equivalente a AES-256).
3. La hibridación para estas firmas es opcional.
4. Para XMSS/LMS, el estado es un dato muy crítico y debe protegerse en integridad.

B.4.2. Modos de hibridación

591. Como se ha mencionado antes, la principal recomendación, mientras no se adopte una solución completamente postcuántica, es hacer uso de un sistema híbrido, al menos a corto y medio plazo. Esta recomendación no es aplicable al caso en el que la seguridad recaiga exclusivamente en funciones hash, siempre que estas tengan las longitudes señaladas.

592. En esta sección trataremos el tema de cómo combinar dos o más mecanismos de intercambio de claves en un acuerdo de claves híbrido, suponiendo que todos los esquemas KEM proporcionen al menos seguridad OW-CPA. En todo caso, es importante tener en cuenta lo que se menciona en la Nota 120.

593. Nota 120 **[No es aceptable un XOR con dos claves]** Llevar a cabo una operación de XOR con dos claves aleatorias no es aceptable porque supone una falta de seguridad importante. Si el tipo de ataque es pasivo, esta operación podría no suponer un problema; pero si el ataque es activo de modo que se pudiera replicar el contenido de un registro en otro, las consecuencias de este ataque podrían ser nefastas.

594. Como es sabido, un mecanismo de intercambio de claves entre dos usuarios, el iniciador A y el respondedor B consta, en general, de tres algoritmos: uno de generación de claves, KeyGen(), que produce una clave privada, sk y una clave

pública pk; un algoritmo de respuesta con entrada la clave pública, Response(pk), que da lugar a un secreto compartido, k, y un valor de respuesta, R o un indicador de error, $\perp$; y un algoritmo de recepción cuyas entradas son la clave privada y la respuesta, Receive(sk, R), que calcula la clave compartida secreta, k, o un indicador de error, $\perp$.

595. Si el algoritmo Response devuelve un indicador de error, B responderá con un mensaje de error y finalizará el proceso. Si A recibe un mensaje de error de B, o si Receive devuelve un indicador de error, A finalizará el proceso.

B.4.2.1. Hibridación *CAT then KDF*

596. El esquema de acuerdo de clave híbrida concatenada con una KDF [ETS20, §8.2], a veces denotado como CatKDF o *CAT then KDF*, permite intercambiar múltiples claves públicas y múltiples valores de respuesta en un solo mensaje. Su construcción se muestra en la Tabla B.13.

Iniciador A		**Respondedor** B
$(sk_1, pk_1) = \mathsf{KeyGen}_1()$		
$(sk_2, pk_2) = \mathsf{KeyGen}_2()$		
$M_A = (pk_1, pk_2, \ldots)$		
	$\xrightarrow{M_A}$	
		(k_1, R_1) o $\perp = \mathsf{Response}_1(pk_1)$
		(k_2, R_2) o $\perp = \mathsf{Response}_2(pk_2)$
		$M_B = (R_1, R_2, \ldots)$ o mensaje de error
	$\xleftarrow{M_B}$	
k_1 o $\perp = \mathsf{Receive}_1(sk_1, R_1)$		
k_2 o $\perp = \mathsf{Receive}_2(sk_2, R_2)$		

Tabla B.13: Acuerdo de clave híbrido concatenado o *CAT then KDF*

597. Debe tenerse en cuenta que si algún algoritmo Response devuelve un indicador de error, B responderá con un mensaje de error y finalizará el proceso. Si A recibe un mensaje de error, A finalizará el proceso. Si algún algoritmo Receive devuelve un indicador de error, A deberá dar por terminado el proceso.

598. Por otra parte, M_A es una cadena de octetos que contiene una codificación de las claves públicas, pk_i, intercambiadas desde el iniciador al respondedor. M_A puede incluir información de negociación de la sesión, si es necesario. En el caso de que se utilicen más de dos sistemas de establecimiento clave, M_A contendrá todas las claves públicas. Por su parte, si M_B no es un mensaje de error, será una cadena de octetos que contendrá una codificación de los valores de respuesta R_i. Además, M_B puede incluir información sobre la negociación de la sesión. En el caso de que

se utilicen más de dos sistemas de establecimiento clave, M_B contendrá todas las claves públicas y textos cifrados correspondientes.

599. El algoritmo *CAT then KDF* hace uso de una función de derivación de claves, KDF, autorizada (ver Tabla 2.11) y una función de contexto, $f_{context}$. Este se muestra como Algoritmo 35.

Algoritmo 35 Hibridación *CAT then KDF*

Entrada: una clave secreta precompartida, psk, si se considera conveniente, en otro caso, será un octeto vacío; una n-tupla de cadenas de octetos que contienen secretos compartidos intercambiados mediante un intercambio de claves híbridas, $(k_1, k_2, \ldots, k_n)$; las cadenas de octetos de un par de mensajes intercambiados en el establecimiento de los secretos compartidos, M_A, M_B; una cadena de octetos establecida por la instancia de la transacción de intercambio de claves, $context$; una cadena de octetos que especifica una separación de uso para la instancia del intercambio de claves, $label$; y la longitud en octetos, len, del material de la clave derivada, key.
Salida: clave derivada, key.

1. $secret \leftarrow (psk||k_1||k_2||\cdots||k_n)$
2. $f_{context} \leftarrow f\,(context, M_A, M_B)$
3. $key = KDF(secret, label, f_{context}, len)$
4. **Devuelve** key

600. El valor $f_{context}$ se obtiene de la función de formato de contexto f, la cual hace uso de una función hash, H, autorizada (ver Tabla 2.2). Dicha función hash tiene unas entradas con un formato específico definido [ETS20, §7.2]. De hecho, $f_{context} = f(val_1, val_2, \ldots)$ toma una secuencia de cadenas de octetos cada uno de longitud menor que 2^{32} octetos y la convierte en una cadena de octetos de longitud delimitada. Esta función de formato de contexto lleva a cabo el siguiente el proceso:

1. buffer $= \emptyset$
2. for $i = 1$ to n
   a) $len \leftarrow len(val_i)$: devuelve la longitud de val_i en octetos
      1) $L_i = [len_i]$: valor entero de 32 bits expresado como 4 octetos
      2) $V_i \leftarrow val_i$
   b) buffer $\leftarrow$ buffer$r||L_i||V_i$
3. $f_{context} \leftarrow H$(buffer)

B.4.2.2. Hibridación *Cascade*

601. Otro protocolo de acuerdo de clave híbrida se denomina en cascada con KDF, CasKDF o simplemente *Cascade* [ETS20, §8.3], cuya construcción se presenta en la Tabla B.14.

Iniciador A		**Respondedor** B
$(sk_1, pk_1) = \mathsf{KeyGen}_1()$		
$M_{A_1} = (pk_1, \ldots)$		
	$\xrightarrow{M_{A_1}}$	
		(k_1, R_1) o $\perp = \mathsf{Response}_1(pk_1)$
		$M_{B_1} = (R_1, \ldots)$ o mensaje de error
	$\xleftarrow{M_{B_1}}$	
k_1 o $\perp = \mathsf{Receive}_1(sk_1, R_1)$		
$(sk_2, pk_2) = \mathsf{KeyGen}_2()$		
$M_{A_2} = (pk_2, \ldots)$		
	$\xrightarrow{M_{A_2}}$	
		(k_2, R_2) o $\perp = \mathsf{Response}_2(pk_2)$
		$M_{B_2} = (R_2, \ldots)$ o mensaje de error
	$\xleftarrow{M_{B_2}}$	
k_2 o $\perp = \mathsf{Receive}_2(sk_2, R_2)$		
	.....	

Tabla B.14: Acuerdo de clave híbrido en cascada o *Cascade*

602. Si algún algoritmo Response devuelve un indicador de error, B devolverá un mensaje de error y finalizará el proceso. Si A recibe un mensaje de error de B, A finalizará el proceso y si algún algoritmo Receive devuelve un indicador de error, A finalizará el proceso.

603. Cada M_{A_i} es una cadena de octetos que contiene una codificación de la clave pública pk_i intercambiada entre el iniciador y el respondedor. Además, M_{A_i} puede incluir información de negociación de sesión. Por su parte, M_{B_i} se una cadena de octetos que contiene y codifica el valor de respuesta R_i. M_{B_i} puede incluir negociación de sesión información. En este modelo en cascada, pueden ocurrir dos o más rondas.

604. Como en el modelo de hibridación *CAT then KDF*, KDF es una función de derivación de claves autorizada (ver Tabla 2.11); mientras que PRF es una

función pseudo-aleatoria, también autorizada (ver Table 4.21) y se presenta como Algoritmo 36.

Algoritmo 36 Hibridación *Cascade*

Entrada: una clave secreta precompartida, psk, de longitud len_{psk}, si se considera conveniente, en otro caso, será un octeto vacío; una n-tupla de cadenas de octetos que contienen secretos compartidos intercambiados mediante un intercambio de claves híbridas, $(k_1, k_2, \ldots, k_n)$; las cadenas de octetos de los mensajes intercambiados en el establecimiento de los secretos compartidos, M_{A_i}, M_{B_i}; cadenas de octetos establecidas por las instancias de la transacción de intercambio de claves secretas, $context_i$; cadenas de octetos que especifican una separación de uso para las instancias del intercambio de claves, $label_i$; y las longitudes en octetos, len_i, del material de las claves derivadas, key_i, $1 \leq i \leq n$.
Salida: una secuencia de claves intermedias y finales, key_i, de longitudes len_i, $1 \leq i \leq n$ y una secuencia de cadenas secretas, $secret_i$.

1. $secret_0 \leftarrow psk$
2. **for** $i = 1, \ldots, n$ **do**
3. $round_secret_i \leftarrow PRF\,(secret_{i-1}, k_i, M_{A_i}, M_{B_i})$
4. $(secret_i || key_i) \leftarrow KDF\,(round_secret_i, label_i, context_i, len_{psk} + len_i)$
5. **end do**
6. **Devuelve** $(secret_1, secret_2, \ldots, secret_n, key_1, key_2, \ldots, key_n)$

C. Esquemas de Firma Digital Basados en el Problema Logaritmo Discreto

605. Algoritmos DSA y ECDSA

606. De forma muy resumida, el algoritmo de firma digital o DSA (*Digital Signature Algorithm*) [FIPS186-4] se muestra como Algoritmo 37.

Algoritmo 37 DSA: generación de parámetros

1. Un primo p y otro primo q, divisor de $p-1$.
2. Un generador g de un subgrupo cíclico de $\mathbb{Z}_p^*$ de orden q.
3. Su clave privada será el entero, x, con $0 < x < q$.
4. Su clave pública se determina calculando $y = g^x \pmod p$.

607. Para la elaboración de la firma digital estándar o DSS (*Digital Signature Standard*) de un mensaje, m, se sigue el Algoritmo 38.

Algoritmo 38 DSA: elaboración de la firma digital

1. Se selecciona un entero aleatorio $k \in \mathbb{Z}_q^*$.
2. Se calcula $r = \left(g^k \pmod p\right) \pmod q$.
3. Se resuelve en la incógnita s la congruencia dada por $m = -x \cdot r + k \cdot s \pmod q$, calculando $k^{-1} \pmod q$ y $s = k^{-1}(m + x \cdot r) \pmod q$.
4. La firma digital es el par (r, s).

608. Para verificar la firma digital (r, s) de m, se sigue el Algoritmo 39.

Algoritmo 39 DSA: verificación de la firma digital

1. Se determina $w = s^{-1} \pmod q$.
2. Se calculan $u_1 = m \cdot w \pmod q$ y $u_2 = r \cdot w \pmod q$.
3. Se determina $v = (g^{u_1} y^{u_2} \pmod p) \pmod q$.
4. Se acepta la firma si y solo si $v = r$.

609. El algoritmo de firma digital basado en curvas elípticas (*Elliptic Curve Digital Signature Algorithm* o ECDSA) fue propuesto originalmente en 1992 por Scott Vanstone en respuesta a la convocatoria del NIST para el establecimiento de un

estándar de firma digital (*Digital Signature Standard* o DSS). Posteriormente, fue aceptado como estándar por ISO [ISO14888-3], ANSI X9.62 [ANSIX9.6], IEEE [IEEE1363a] y NIST ([FIPS186-4]).

610. Como todos los esquemas de firma digital, tiene tres fases: 1) generación de claves, 2) elaboración de la firma y 3) verificación de la firma. Los parámetros del ECDSA son una curva elíptica definida sobre un cuerpo finito primo, $\mathbb{F}_p$, $E(\mathbb{F}_p)$ y un punto de la curva de orden primo q, $G \in E$, que actúa como generador con $q \approx p$.

611. La clave privada del usuario $\mathcal{A}$ es un entero aleatorio en el intervalo $[1, q-1]$, a, y su clave pública es el punto de la curva dado por $A = aG$.

612. Para elaborar la firma del documento m, $\mathcal{A}$ sigue el Algoritmo 40.

Algoritmo 40 ECDSA: elaboración de la firma digital

1. Genera al azar un entero k, $1 < k < q-1$. Calcula $kG = (x_1, y_1)$ y $r = x_1 \pmod q$. Si $r = 0$ elige otro valor de k.
2. Calcula $s = k^{-1}(\tilde{m} + a \cdot r) \pmod q$, siendo $\tilde{m}$ el resumen de m por una función resumen segura acordada de antemano. Si $s = 0$, se elige otro valor para k.
3. La firma para m de $\mathcal{A}$ es el par (r, s).

613. Para verificar la firma de $\mathcal{A}$ para m, el usuario $\mathcal{B}$ procede según se indica en el Algoritmo 41.

Algoritmo 41 ECDSA: verificación de la firma digital

1. Obtiene los parámetros y la firma de $\mathcal{A}$, (r, s) para m.
2. Comprueba que $r, s \in [1, q-1]$.
3. Calcula $w = s^{-1} \pmod q$ y $\tilde{m}$.
4. Determina $u_1 = \tilde{m} \cdot w \pmod q$ y $u_2 = r \cdot w \pmod q$.
5. Calcula $u_1 G + u_2 A = (x_0, y_0)$ y $v = x_0 \pmod q$.
6. Acepta la firma si y solo si se cumple que $v = r$.

614. Algoritmos de firma de Schnorr y EC-Schnorr

615. La firma propuesta por Schnorr es una variante del esquema de firma de DSA [Sch91]. Este esquema usa el mismo protocolo de generación de claves que el DSA sin restricciones en el tamaño de los parámetros y también emplea un subgrupo de orden q de $\mathbb{Z}_p^*$. Además, se emplea una función resumen $\mathfrak{h}\colon \{0,1\}^* \to \mathbb{Z}_q^*$.

616. Los pasos a seguir para la elaboración de la firma para un mensaje m son los mostrados en el Algoritmo 42.

Algoritmo 42 Schnorr: elaboración de la firma digital

1. El firmante genera un entero aleatorio secreto k, con $1 \leq k \leq q$.
2. Calcula $u = g^k \pmod p$, $r = \mathfrak{h}(m||u)$ y $s = (ar + k) \pmod q$, siendo a su clave privada.
3. La firma es el par (s, r).

617. Para la verificación de la firma se sigue el Algoritmo 43.

Algoritmo 43 Schnorr: verificación de la firma digital

1. El verificador calcula $w = g^s A^{-r} \pmod p$ y $\bar{r} = \mathfrak{h}(m||w)$, donde $A = g^a \pmod p$ es la clave pública del firmante.
2. La firma se acepta si y solo si $r = \bar{r}$.

618. La versión de la firma de Schnorr para curvas elípticas se conoce como EC-Schnorr (*Elliptic Curve Based Schnorr Signature Algorithm*) [TR-03111v2.1].

619. En este algoritmo se considera que la clave privada del firmante, $\mathcal{A}$ es d_A y su clave pública es A, siendo (p, a, b, G, n, h) los parámetros de la curva elíptica E, esto es, $p > 2$ es un primo que define el cuerpo base, a, b los parámetros de la curva, G un punto de la curva elíptica de orden n y h el cofactor correspondiente. El algoritmo para la elaboración de la firma es el Algoritmo 44.

Algoritmo 44 EC-Schnorr: elaboración de la firma digital

1. El firmante genera un entero aleatorio secreto k, con $1 \leq k \leq n - 1$.
2. Calcula $Q = kG = (Q_x, Q_y)$ en E y $r = \mathrm{OS2I}(\mathrm{FE2OS}(Q_x)) \pmod n$. Si $r = 0$, se elige otro valor para k.
3. Determina el valor de $s = (kr - \mathrm{OS2I}(\langle(m)) d_A \pmod n$. Si $s = 0$, se elige otro valor para k.
4. La firma es el par (r, s).

620. En el algoritmo anterior, $\mathrm{OS2I}$ es una primitiva que convierte cadenas de octetos en cadenas de enteros (*Octet String to Integer Conversion*) como sigue: si $o_{l-1} o_{l-2} \ldots o_2 o_1$ es una cadena de l octetos, cada uno de ellos se puede interpretar

como un entero no negativo en base 256, de modo que el bit más significativo es el que está más a la izquierda; en definitiva, se obtiene el entero dado por

$$x = x_{l-1} \cdot 256^{l-1} + x_{l-2} \cdot 256^{l-2} + \cdots + x_1 \cdot 256 + x_0, \qquad \text{(C.1)}$$

con $0 \leq i \leq l-1$. Por su parte, la primitiva $\mathrm{FE2OS}$ convierte elementos del cuerpo base en cadenas de octetos (*Field Element to Octet String Conversion Primitive*). En este caso, si $x \in \mathbb{F}_p$ es un elemento del cuerpo base, se convierte en una cadena de octetos de longitud $l = \lceil \log_{256} p \rceil$ aplicando la función de conversión $\mathrm{I2OS}$ con el parámetro l, es decir,

$$\mathrm{FE2OS}(x) = \mathrm{I2OS}(x, l).$$

Finalmente, la primitiva que convierte enteros en cadenas de octetos, I2SO (*Integer to Octet String Conversion Primitive*), considera un entero no negativo, x, con una longitud deseada, l, y lo convierte en una cadena de octetos. El parámetro l debe verificar $256^l > x$. La idea es escribir el entero x en su representación única como un l-dígito en base 256 como se muestra en la expresión (C.1), con $x_i < 256$ para $0 \leq i \leq l-1$, de modo que la cadena de octetos es $o_{l-1}o_{l-2}\ldots o_2 o_1$.

621. El algoritmo de verificación de la firma de EC-Schnorr es el Algoritmo 45.

Algoritmo 45 EC-Schnorr: verificación de la firma digital

1. El verificador comprueba los tamaños de r y s, esto es, $r, s \in \{1, 2, \ldots, n-1\}$.
2. Calcula $\bar{r} = r^{-1} \pmod{n}$
3. Determina los valores de $u_1 = \bar{r} \cdot \mathrm{OS2I}(\mathfrak{h}(m)) \pmod{n}$ y $u_2 = \bar{r}s \pmod{n}$.
4. Calcula $Q = (u_1 G + u_2 A)$ sobre E. Si $Q = \mathcal{O}$, el algoritmo finaliza con error.
5. Calcula $v = \mathrm{OS2I}(\mathrm{FE2OS}(Q_x)) \pmod{n}$.
6. Acepta la firma si y solo si $v = r$.

622. **Algoritmo EdDSA**

623. En la sección 3.1.3 se presentó la expresión general de una curva elíptica definida sobre un cuerpo finito, $\mathbb{F}$. Allí se mencionó que dependiendo del cuerpo considerado, tal curva puede tener expresiones más sencillas.

624. Las expresiones de curvas elípticas más extendidas son las denominadas curvas de Weierstrass, cuya forma es la siguiente:

$$E : y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6, \text{ donde } a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}.$$

625. En el caso particular de que el cuerpo finito sea un cuerpo primo, $\mathbb{F}_p$, de característica diferente de 2 y de 3, mediante un adecuado cambio de variables, la expresión anterior se transforma en lo que se conoce como expresión reducida de Weierstrass:

$$E : y^2 = x^3 + ax + b \text{ donde } a, b, c \in \mathbb{F}, \text{ verificando } -16\left(4a^3 + 27b^2\right) \neq 0.$$

626. Además de las curvas de Weierstrass mencionadas, existen otros dos tipos de curvas especiales definidas sobre cuerpos primos, que son las llamadas curvas de Edwards y curvas de Montgomery. De los dos tipos de curvas, nos detendremos en las curvas de Edwards.

627. Estas curvas fueron introducidas por Harold M. Edwards [Edw07] como aquellas curvas elípticas que responden a la siguiente ecuación, conocida como forma normal:

$$x^2 + y^2 = c^2(1 + x^2y^2), \text{ siendo } c \text{ una constante.}$$

628. Más tarde, con el fin de aumentar el número de curvas elípticas que pudieran transformarse en curvas de Edwards sin modificar el cuerpo base original, Bernstein y Lange [BL07], diseñaron una variante de estas curvas, denominada forma generalizada, cuya ecuación tiene alguna de las dos siguientes expresiones (que son isomorfas):

$$x^2 + y^2 = c^2(1 + dx^2y^2), \text{ donde } cd(1 - dc^4) \neq 0 \pmod{p},$$
$$x^2 + y^2 = 1 + dx^2y^2.$$

629. Posteriormente, en [BBJ+08] se propuso otra generalización de las curvas de Edwards, conocidas como curvas *twisted* (torcidas) de Edwards, y que corresponden a la siguiente expresión:

$$ax^2 + y^2 = 1 + dx^2y^2.$$

630. Una vez presentadas las curvas twisted de Edwards, abordaremos el algoritmo de firma digital que emplea estas curvas y que se conoce como EdDSA (ver [FIPS186-5] y [RFC8032]). Este esquema está basado en el de firma de Schnorr, que emplea curvas twisted de Edwards. En [SP800-186] se listan las curvas aprobadas para su uso con EdDSA.

631. El esquema de firma EdDSA consta de dos procesos previos antes de la generación de claves, elaboración de la firma y verificación de la misma. Estos dos procesos se conocen como Codificación y Decodificación y pasamos a presentarlos a continuación.

632. Los parámetros empleados en el esquema EdDSA se codifican como cadenas de octetos mientras que los números enteros se codifican utilizando la convención *little-endian*, es decir, la cadena de 32 octetos dada por $h = h_0, h_1, \ldots, h_{31}$ representa al entero dado por $h_0 + 2^8h_1 + \cdots + 2^{248}h_{31}$. En esta expresión, el byte más significativo es h_{31} y el menos significativo es h_0.

633. Por otra parte, para el punto de curva (x, y) con $0 \leq x, y < p$, en primer lugar se codifica la coordenada y como una cadena little-endian de 32 octetos para la curva Ed25519 o de 57 octetos para la curva Ed448. El bit más significativo del octeto final de Ed25519 es cero, mientras que para Ed448, es cero el octeto más significativo. Para formar la codificación del punto, se copia el bit menos significativo de la coordenada x al bit más significativo del octeto final.

634. La decodificación de un punto dado como una cadena de 32 octetos es algo más complicado. Para ello se sigue el siguiente proceso:

    1. Se considera la cadena de octetos como un número entero en representación little-endian. El bit más significativo de este entero es el bit menos significativo de la coordenada x, denotada como x_0. La coordenada y se recupera simplemente borrando este bit. Si el valor resultante es $\geq p$, la decodificación es errónea.

    2. Para recuperar la coordenada x, la ecuación de la curva implica que $x^2 = \dfrac{y^2 - 1}{dy^2 - a} \pmod p$ (el denominador siempre es distinto de 0 modulo p). A continuación se calcula una raíz cuadrada para obtener x; cómputo que puede hacerse siguiendo el algoritmo de Tonelli-Shanks [SP800-186, Appendix E].

635. Las claves públicas del esquema EdDSA tienen exactamente b bits y las firmas $2b$ bits. El valor b es múltiplo de 8, por lo que la longitud de la clave pública y de la firma son un número entero de octetos.

636. Para la curva Ed25519, se toma $b = 256$, por lo que la clave privada debe tener 32 octetos. Por su parte, para la curva Ed448, es $b = 456$ y la clave privada tiene 57 octetos [RFC8032].

637. El algoritmo para generar el par de claves privada (cadena de b bits) y pública (punto codificado de la curva) se presenta como el Algoritmo 46 [FIPS186-5, Appendix A].

Algoritmo 46 EdDSA: generación de claves

1. Se genera una cadena de b bits aleatoria mediante un generador determinista de bits aleatorios autorizado (ver el Capítulo 5), que será la clave privada k.

2. Se calcula el hash de la clave privada obtenida: $\mathfrak{h}(k) = (h_0, h_1, \ldots, h_{2b-1})$ haciendo uso de la función SHA-512 para la curva Ed25519 y de SHAKE-256 para la curva Ed448, esto es, $\mathfrak{h}(k) = \text{SHAKE-256}(k, 912)$.

3. La primera mitad de $\mathfrak{h}(k)$, es decir, la cadena $hres_1 = (h_0, h_1, \ldots, h_{b-1})$ se utiliza para generar la clave pública. Esta cadena se modifica para cada tipo de curva de la siguiente manera:

   a) Para las curvas Ed25519, los primeros tres bits del primer octeto y el último bit del último octeto se ponen a cero y el penúltimo bit del último octeto se pone a uno. Es decir, $h_0 = h_1 = h_2 = h_{b-1} = 0, h_{b-2} = 1$.

   b) Para las curvas Ed448, los dos primeros bits del primer octeto y los ocho bits del último octeto se ponen a cero y el último bit del penúltimo octeto se pone a uno. Esto es, $h_0 = h_1 = h_i = 0,$ para $b-8 \leq i \leq b-1, h_{b-9} = 1$.

4. Se calcular un número entero s a partir del valor de $hres_1$ usando el convenio little-endian.

5. Se calcula el punto sG, de modo que la clave pública EdDSA es el punto Q correspondiente a la codificación de sG, siendo G el generador de la curva considerado.

638. Para elaborar la firma EdDSA de un mensaje M se sigue el Algoritmo 47.

Algoritmo 47 EdDSA: elaboración de la firma digital

Entrada: la cadena de bits a firmar, M, el par de claves pública-privada, (Q, k), el orden del subgrupo mayor de la curva considerada, n, y una función hash, $\mathfrak{h}$. Para Ed448, una cadena, $context$, establecida por el firmante y el verificador con una longitud máxima de 255 octetos (por defecto, la cadena es vacía).
Salida: la firma, $R||S$, siendo R la codificación de un punto y S una cadena de octetos de una longitud determinada de un valor codificado en little-endian.

1. Calcular el hash de la clave privada, $\mathfrak{h}(k) = (h_0, h_1, \ldots, h_{2b-1})$, siendo SHA-512 para Ed25519 y SHAKE256 para Ed448 ($\mathfrak{h}(k) =$ SHAKE-256$(k, 912)$). El valor de $\mathfrak{h}(k)$ se puede calcular con antelación.

2. Utilizando la segunda mitad de $\mathfrak{h}(k)$, esto es, $hres_2 = (h_b, h_{b+1}, \ldots, h_{2b-1})$, se define:

   a) $r =$ SHA-512$(hres_2||M)$, para curvas Ed25519 y se interpreta r como un entero little-endian de 64 octetos.

   b) $r =$ SHAKE-256$(dom4(0, context)||hres_2||m, 912)$, para curvas Ed448. El valor $dom4(f, c)$ se define en [RFC8032] como ("SigEd448"||octet(f)||octet(octetlength(c))||c). La cadena "SigEd448" está en ASCII (8 octetos). El valor octet(f) es el octeto con valor f y octetlength(c) es el número de octetos de la cadena c. Se interpreta r como un entero little-endian de 114 octetos.

3. Calcular el punto rG. La cadena de octetos R es la codificación de dicho punto.

4. Determinar s a partir de $\mathfrak{h}(k)$ como en el proceso de generación de pares de claves. Para ello se emplean las cadenas de octetos R, Q y M para definir:

   a) $digest =$ SHA-512$(R||Q||M)$, en el caso de curvas Ed25519.

   b) $digest =$ SHAKE-256$(dom4(0, context)||R||Q||M, 912)$, para curvas Ed448.

   c) Se interpreta $digest$ como un entero little-endian.

5. Calcular $S = (r + digest \cdot s) \pmod n$. La cadena de octetos S es la codificación del número entero resultante.

6. La firma es la concatenación de las cadenas de octetos R y S: $R||S$.

639. Los pasos para verificar la firma EdDSA son los del Algoritmo 48.

Algoritmo 48 EdDSA: verificación de la firma digital

Entrada: el mensaje a verificar, M, la firma, $R||S$, siendo R y S cadenas de octetos y la clave pública Q. Para Ed448, una cadena, $context$, establecida por el firmante y el verificador con una longitud máxima de 255 octetos (por defecto, la cadena es vacía).
Salida: aceptación o rechazo de la validez de la firma.

1. Decodificar la primera mitad de la firma como un punto R y la segunda mitad de la firma como un número entero t. Comprobar que el entero t está en el rango $0 \leq t < n$. Decodificar la clave pública Q obteniendo el punto Q'. Si alguna de las decodificaciones falla, la salida es "rechazar".

2. Usando la función hash establecida o una función con salida extensible (Extendable-Output Function o XOF),

   a) Calcular $digest = \text{SHA-512}(R||Q||M)$ para Ed25519.

   b) Calcular $digest = \text{SHAKE-256}(dom4(0, context)||R||Q||M, 912)$ para Ed448.

   c) Interpretar $digest$ como un entero little-endian u.

3. Comprobar que se cumple la ecuación de verificación $2^c tG = 2^c R + 2^c uQ'$. Basta con comprobar (aunque no es obligatorio) que $tG = R + uQ'$. La salida es "rechazar" si falla la verificación; en caso contrario la salida es "aceptar".

640. En la Tabla C.1 se muestran los principales parámetros para las curvas twisted de Edwards aprobadas para su uso en EdDSA.

Curva	Seguridad	Parámetros: (p, a, c, d)	Referencias	Notas
Ed25519	128	$(2^{255} - 19, -1, 3, \frac{-121665}{121666})$	[FIPS186-5], [SP800-186], [RFC8032]	121, 122
Ed448	224	$(2^{448} - 2^{224} - 1, 1, 2, -39081)$	[FIPS186-5], [SP800-186], [RFC8032]	121, 123

Tabla C.1: Curvas twisted de Edwards aprobadas para su uso en el esquema EdDSA

641. Nota 121 [**Firmas EdDSA deterministas**] La firmas EdDSA son deterministas dado que en el proceso de generación de firma se emplea un valor único calculado a partir del hash de la clave privada y del mensaje. Este proceso protege las firmas contra ataques que aprovechan el proceso de generación de firmas con una aleatoriedad insuficiente.

642. Nota 122 [**Hash con Edwards25519**] El esquema de firma EdDSA con la curva Edwards25519 utilizará la función SHA-512.

643. Nota 123 [**Hash con Edwards448**] El esquema de firma EdDSA con la curva Edwards448 utilizará la función SHAKE-256 (ver [FIPS202]).

644. **Algoritmo HashEdDSA**

645. El esquema de firma digital conocido como Prehash basado en curvas de Edwards o HashEdDSA es una versión del esquema de firma digital EdDSA. La principal diferencia entre ambos esquemas es que HashEdDSA genera una firma en el hash del mensaje M; mientras que EdDSA firma el mensaje M directamente. Los parámetros y la generación de claves de ambos esquemas son los mismos, si bien para HashEdDSA se consideran dos opciones, denominadas Ed25519ph y Ed448ph [FIPS186-5].

646. El proceso que debe seguirse para elaborar la firma HashEdDSA para un mensaje M se presenta en el Algoritmo 49.

Algoritmo 49 HashEdDSA: elaboración de la firma digital

Entrada: la cadena de bits a firmar, M, el par de claves pública-privada, (Q, k), el orden del subgrupo mayor de la curva considerada, n, y una función hash, $\mathfrak{h}$, que será SHA-512 para curvas Ed25519ph y SHAKE-256 para curvas Ed448ph. Una cadena, $context$, establecida por el firmante y el verificador con una longitud máxima de 255 octetos (por defecto, la cadena es vacía).
Salida: la firma, $R||S$, siendo R la codificación de un punto y S un valor codificado en little-endian.

1. Calcular $\mathfrak{h}(M) = \text{SHA-512}(M)$ para Ed25519ph o $\mathfrak{h}(M) = \text{SHAKE-256}(M, 512)$ para Ed448ph.
2. Calcular el hash de la clave privada k, $\mathfrak{h}(k) = (h_0, h_1, \ldots, h_{2b-1})$ usando SHA-512 para Ed25519ph y SHAKE-256 para Ed448ph ($\mathfrak{h}(k) = \text{SHAKE-256}(k, 912)$). El valor de $\mathfrak{h}(k)$ se puede calcular con antelación.
3. Usando la segunda mitad del resumen $hres_2 = h_b|| \cdots ||h_{2b-1}$, definir
   a) $r = \text{SHA-512}(dom2(1, context)||hres_2||\mathfrak{h}(M))$, con 64 octetos. El valor de $dom2(f, c)$ se define como la cadena de octetos ("SigEd25519 no Ed25519 collisions"||octet(f)||octet(octetlength(c))||c). La cadena "SigEd25519 no Ed25519 collisions" está en ASCII (32 octetos). El valor octeto(f) es el octeto con valor f, y longitud del octeto(c) es el número de octetos en la cadena c. El valor de la cadena $context$ lo establecen el firmante y el verificador (255 octetos, como máximo), siendo la cadena vacía el valor por defecto, para Ed25519ph.
   b) $r = \text{SHAKE-256}(dom4(1, contexto)||hres_2||\mathfrak{h}(M), 912)$, donde el firmante y el verificador establecen $context$ (255 octetos, como máximo), donde la cadena vacía es el valor por defecto, para curvas Ed448ph.
   c) Interpretar r como un entero little-endian.
4. Calcular el punto rG. La cadena de octetos R es la codificación de rG.
5. Utilizar las cadenas de octetos R, Q y $\mathfrak{M}$ para definir
   a) $digest = \text{SHA-512}(dom2(1, context)||R||Q||\mathfrak{h}(M))$, con Ed25519ph.
   b) $digest = \text{SHAKE-256}(dom4(1, context)||R||Q||\mathfrak{h}(M), 912)$, con Ed448ph.
   c) Interpretar $digest$ como un entero little-endian.
6. Calcular $S = (r + digest \cdot s) \pmod n$. La cadena de octetos S es la codificación del entero resultante.
7. La firma resultante es la concatenación de los octetos R y S: $R||S$.

647. Los pasos a seguir para verificar la firma HashEdDSA se muestran en el Algoritmo 50.

Algoritmo 50 HEdDSA: verificación de la firma digital

Entrada: el mensaje a verificar, M, la firma, $R||S$, siendo R y S cadenas de octetos, el orden del subgrupo mayor de la curva considerada, n, y la clave pública Q. Una cadena, $context$, establecida por el firmante y el verificador con una longitud máxima de 255 octetos (por defecto, la cadena es vacía).
Salida: aceptación o rechazo de la validez de la firma.

1. Decodificar la primera mitad de la firma como un punto R y la segunda mitad de la firma como un número entero s.

2. Verificar que el entero s está en el rango $0 \leq s < n$. Decodificar la clave pública Q como un punto Q'. Si alguna de las decodificaciones falla, la salida es "rechazar".

3. Construir la cadena de bits $HashData$ como la concatenación de las cadenas de octetos R, Q y $\mathfrak{h}(M)$, esto es, $HashData = R||Q||\mathfrak{h}(M)$, siendo $\mathfrak{h}(M) =$ SHA-512(M) para las curvas Ed25519ph y $\mathfrak{h}(M) =$ SHAKE-256$(M, 512)$ para las curvas Ed448ph.

4. Utilizar las funciones SHA-512 y SHKE-256

   a) Calcular $digest =$ SHA-512$(dom2(1, context)||HashData$ para Ed25519ph.

   b) Calcular $digest =$ SHAKE-256$(dom4(1, context)||HashData, 912)$ para Ed448ph.

5. Comprobar que se verifica la ecuación de verificación $2^c sG = 2^c R + 2^c tQ'$. Basta con comprobar (aunque no es obligatorio) que $sG = R + tQ'$. La salida es "rechazar" si falla la verificación; en caso contrario la salida es "aceptar".

648. Así pues, EdDSA aplica el hash al mensaje dos veces, mientras que HashEdDSA solo lo hace una vez. Otra diferencia entre ambos esquemas es que EdDSA almacena en el buffer el mensaje completo (o leerse dos veces desde su almacenamiento). Este hecho supone que para mensajes largos, HashEdDSA debería tener un rendimiento mejor.

649. Por otra parte, si fuera factible calcular colisiones en la función hash (o XOF) utilizada, no parece que esto suponga ningún efecto adverso en la seguridad de EdDSA. Sin embargo, esta propiedad no es válida para HashEdDSA dado que las colisiones pueden dar lugar a mensajes falsificados.

650. Algoritmos KCDSA y EC-KCDSA

651. Existen algunas variantes de los algoritmos DSA y ECDSA que pueden verse en [SJP14]. En particular, unas breves descripciones de las variantes KCDSA, EC-KCDSA y EC-GDSA se incluyen a continuación.

652. En [KCD98, LL99], se ha propuesto un algoritmo como estándar para la firma digital coreana basado en certificados (*Korean Certificate-based Digital Signature Algorithm* o KCDSA). En este algoritmo de firma, la clave pública se valida mediante un certificado del tipo X-509 emitido por una autoridad de confianza.

653. Los parámetros del dominio del KCDSA, esto es, los parámetros que comparten un grupo de usuarios son los siguientes:

- Un primo grande, p, de modo que su tamaño (longitud en bits) varíe entre 512 y 2048 bits, es decir, $|p| = 512 + 256i$, $0 \leq i \leq 6$, con incrementos de múltiplos de 256 bits.
- Un factor primo q de $p-1$ tal que $|q| = 128 + 32j$, $1 \leq j \leq 4$. Esto es, el tamaño de q puede variar entre 128 y 256 bits, con incrementos de múltiplos de 32 bits. Además, se requiere que $(p-1)/2q$ sea primo o, al menos, que todos sus factores primos sean mayores que q. La razón de esta última restricción es evitar los ataques contra los subgrupos de orden pequeño de $\mathbb{Z}_p^*$.
- Un elemento base, g, de orden q modulo p, es decir, $g \neq 1$ and $g^q \equiv 1 \pmod{p}$.

654. Por su parte, los parámetros del usuario son x, y, z, generados de la siguiente manera:

- x es la clave privada del firmante, elegida al azar en $\mathbb{Z}_q^*$.
- y es a clave pública del firmante que permite la verificación de su firma y se calcula como $y = g^{x^{-1}} \pmod{p}$, siendo x^{-1} el inverso multiplicativo de x módulo q.
- z es un resumen del valor C_D, es decir, $z = \mathfrak{h}(C_D)$. El valor C_D denota los datos de certificación del firmante que, al menos, debe contener el identificador distinguido del firmante, la clave pública y y los parámetros del dominio $\{p, q, g\}$.

655. El algoritmo para que el firmante, $\mathcal{A}$, elabore su firma digital para el mensaje m es el Algoritmo 51.

Algoritmo 51 KCDSA: elaboración de la firma digital

1. $\mathcal{A}$ genera al azar un entero $k \in \mathbb{Z}_q^*$ y calcula $w = g^k \pmod p$.
2. Calcula $r = \mathfrak{h}(w)$ y $e = r \oplus \mathfrak{h}(z||m) \pmod q$.
3. Determina $s = x(k - e) \pmod q$.
4. (opcional) Si $s = 0$ se elige otro valor de k, si no es así, la firma para m es $(r||s)$.

656. A pesar de que la probabilidad de que $s = 0$ sea despreciable, es conveniente verificar si se obtiene ese valor o no, dado que si $s = 0$, el resultado de la firma es independiente de la clave privada, x, del firmante, lo que puede dar lugar a problemas de autenticación y no repudio. En cualquier caso, no hay peligro de que los valores de x o de k sean descubiertos por un adversario.

657. También es importante señalar que como el paso que lleva más tiempo de computación es la determinación de w, es posible calcular el valor del par (k, r) de forma previa e independientemente del mensaje firmar, lo que puede acelerar los cómputos en línea. En efecto, bastaría entonces con calcular los siguientes dos valores para elaborar la firma digital del mensaje m:

$$r = \mathfrak{h}\left(g^k \pmod p\right), \text{ con } k \in_r \mathbb{Z}_q^*,$$
$$s = x(k - r \oplus \mathfrak{h}(z||m) \pmod q).$$

658. El algoritmo para que el usuario $\mathcal{B}$ verifique la firma $(r||s)$ de $\mathcal{A}$ para el mensaje m es el Algoritmo 52.

Algoritmo 52 KCDSA: verificación de la firma digital

1. En primer lugar comprueba la validez del certificado del firmante, extrae los datos de certificación, C_D, del certificado y calcula su resumen: $z = \mathfrak{h}(C_D)$.
2. Comprueba el tamaño de r y de s, de modo que $0 \leq r < 2^{|q|}$ y $0 < s < q$.
3. Calcula el valor de $e = r\mathfrak{h}(z||m) \pmod q$.
4. Determina $w = y^s g^e \pmod p$.
5. Acepta la firma si y solo si se cumple que $r = \mathfrak{h}(w)$.

659. La variante del KCDSA para curvas elípticas se conoce como EC-KCDSA (*Elliptic Curve Korean Certificate-based Digital Signature Algorithm*).

660. Al igual que en el caso del KCDSA, el EC-KCDSA utiliza un certificado del usuario que va a firmar el mensaje, de modo que los datos de certificado se denotan como

C_D y se utilizan para calcular el valor $z = \mathfrak{h}(C_D)$, siendo $\mathfrak{h}$ la función resumen segura seleccionada. En este caso, C_D debe contener, al menos, el identificador del usuario, su clave pública y los parámetros del dominio.

661. Los parámetros del dominio de EC-KCDSA son los necesarios para definir la curva elíptica sobre el cuerpo que se determine. En este caso, tales parámetros son:

- Un entero positivo, $m > 1$, o un número primo $p > 2$ y un entero m, que definen el cuerpo finito a considerar, $\mathbb{F}_{2^m}$ o $\mathbb{F}_{p^m}$.
- Los coeficientes a, b que definen la curva elíptica, E, dada por su ecuación correspondiente:

$$E: \begin{cases} y^2 + xy = x^3 + ax^2 + b, & \text{para } \mathbb{F}_{2^m} \\ y^2 = x^3 + ax + b, & \text{para } \mathbb{F}_{p^m} \end{cases}$$

- Un primo q que divida al orden de la curva, $\#(E)$. En este caso, la elección de q debe seguir las especificaciones estándares de modo que su tamaño no ponga en peligro la seguridad del sistema.
- Un punto de la curva de orden q, $G = (G_x, G_y)$.

662. Por otra parte, los parámetros del usuario son:

- La clave de firma privada del firmante, elegida al azar, $x \in \mathbb{Z}_q^*$.
- La clave pública de verificación de la firma, Y, calculada como el punto de la curva dado por $Y = x^{-1}G$, siendo x^{-1} el inverso multiplicativo de x módulo q.
- Los datos de certificación, z, como en el esquema KCDSA.

663. El algoritmo para la elaboración de la firma es el Algoritmo 53.

Algoritmo 53 ECKCDSA: elaboración de la firma digital

1. $\mathcal{A}$ genera al azar un entero $k \in \mathbb{Z}_q^*$ y calcula $W = kG$ sobre E.
2. Calcula $r = \mathfrak{h}(W) = \mathfrak{h}(W_x||W_y)$ y $e = r \oplus \mathfrak{h}(z||m) \pmod q$.
3. Determina $s = x(k - e) \pmod q$.

664. Como en el caso del KCDSA, una parte del algoritmo se puede realizar con antelación (fuera de línea), esto es, es posible calcular los valores de

$$r = \mathfrak{h}(kG), \text{ con } k \in_r \mathbb{Z}_q^*,$$
$$s = x(k - r \oplus \mathfrak{h}(z||m) \pmod q).$$

665. El algoritmo de verificación de la firma se presenta como Algoritmo 54.

Algoritmo 54 ECKCDSA: verificación de la firma digital

1. El verificador comprueba la validez del certificado del firmante y calcula su resumen: $z = \mathfrak{h}(C_D)$.
2. Comprueba los tamaños de r y s: $0 \leq r < 2^{|q|}$ y $0 < s < q$.
3. Determina el valor de $e = r\mathfrak{h}(z||m) \pmod q$.
4. Calcula $W = (sY + eG)$ sobre E.
5. Acepta la firma si y solo si $r = \mathfrak{h}(W)$.

666. **Algoritmo EC-GDSA**

667. El esquema EC-GDSA es el algoritmo de firma digital alemán (*Elliptic Curve German Digital Signature Algorithm* [TR-03111v2.1, §4.2.2]. Se sabe que una de las desventajas del esquema ECDSA es que requiere calcular inversos en la fase de elaboración de la firma y teniendo en cuenta que esta operación es una de las más caras en la aritmética modular, evitar tener que hacerla reduce costes y tiempo. Por esta razón, en el esquema EC-GDSA el cálculo del inverso se hace en el momento de la generación de la clave y no cuando se firma un mensaje. Esta modificación tiene en cuenta, además, que la firma se realiza con más frecuencia que la generación de claves, por lo que esta última se mantendrá constante durante un largo periodo de tiempo.

668. Para determinar la clave privada del usuario $\mathcal{A}$, se genera un entero aleatorio en el intervalo $[1, q-1]$, a_0, y se calcula $a = a^{-1} \pmod q$. Su clave pública es el punto de la curva dado por $A = aG$.

669. En la elaboración de la firma del documento m, $\mathcal{A}$ sigue los pasos señalados en el Algoritmo 55.

Algoritmo 55 ECGDSA: elaboración de la firma digital

1. Genera al azar un entero k, $1 < k < q - 1$. Calcula $kG = (x_1, y_1)$ y $r = x_1 \pmod q$. Si $r = 0$ elige otro valor de k.
2. Calcula $s = (k \cdot r - \tilde{m})a \pmod q$, siendo $\tilde{m}$ el resumen de m por una función resumen segura acordada de antemano. Si $s = 0$, se elige otro valor para k.
3. La firma para m de $\mathcal{A}$ es el par (r, s).

670. Para verificar la firma de $\mathcal{A}$ para m, el usuario $\mathcal{B}$ procede como se indica en el Algoritmo 56.

Algoritmo 56 ECGDSA: verificación de la firma digital

1. Obtiene los parámetros y la firma de $\mathcal{A}$, (r, s) para m.
2. Comprueba que $r, s \in [1, q-1]$.
3. Calcula $w = r^{-1} \pmod{q}$ y $\tilde{m}$.
4. Determina $u_1 = \tilde{m} \cdot w \pmod{q}$ y $u_2 = s \cdot w \pmod{q}$.
5. Calcula $u_1 G + u_2 A = (x_0, y_0)$ y $v = x_0 \pmod{q}$.
6. Acepta la firma si y solo si se cumple que $v = r$.

Glosario

0-RTT	Datos tiempo cero de ida y vuelta *zero Round-Trip Time data*
AE	Cifrado autenticado *Authenticated Encryption*
AEAD	Cifrado autenticado con datos asociados *Authenticated Encryption with Associated Data*
AES	Estándar de cifrado avanzado *Advanced Encryption Standard*
AH	Encabezado de autenticación *Authentication Header*
ANSI	*American National Standards Institute*
ANSSI	*Agence Nationale de la Sécurité des Systèmes d'Information*
BIKE	*Bit Flipping Key Encapsulation*
CBC	Encadenamiento de bloques de cifrado *Cipher-Block Chaining*
CBC-CS	Encadenamiento de bloques de cifrado con robo de texto cifrado *Cipher-Block Chaining-Ciphertext stealing*
CC	Criterios Comunes *Common Criteria*
CCA	Ataque al texto cifrado elegido *Chosen Ciphertext Attack*
CCM	Contador con encadenamiento de bloques de cifrado-MAC *Counter with Cipher Block Chaining-Message Authentication Code*
CCN	Centro Criptológico Nacional
CFB	Realimentación del texto cifrado *Cipher Feedback Mode*
CIA	Confidencialidad, Integridad y Autenticidad
CMAC	MAC basado en cifrado *Cipher-based MAC*
CNI	Centro Nacional de Inteligencia
CRYSTALS	*CRYptographic SuiTe for Algebraic LatticeS*
CTR	Contador *Counter Mode*
CVP	Problema del vector más cercano *Closest Vector Problem*
DEM	Mecanismo de encapsulamiento de datos *Data Encapsulation Mechanism*
DES	Cifrado de datos estándar *Data Encryption Standard*
DH	Diffie-Hellman
DHE	Diffie-Hellman con clave efímera

DHC	Diffie-Hellman con cofactor *Diffie-Hellman with cofactor*
DHAES	*Diffie-Hellman Augmented Encryption Scheme*
DHIES	*Diffie-Hellman Integrated Encryption Scheme*
DLAES	*Discrete Logarithm Augmented Encryption Scheme*
DLIES	Esquema de cifrado integrado basado en el logaritmo discreto *Discrete Logarithm Integrated Encryption Scheme*
DLP	Problema del logaritmo discreto *Discrete Logarithm Problem*
DPKE	Cifrado de clave pública determinístico *Deterministic Public Key Encryption*
DSA	Algoritmo de firma digital *Digital Signature Algorithm*
DSS	Estándar de firma digital *Digital Signature Standard*
DRNG	Generador determinista de números aleatorios *Deterministic Random Number Generator*
EAX	Cifrar-luego-autenticar-luego-traducir *Encrypt-then-Authenticate-then-Translate*
EC-DH	Diffie-Hellman basado en curvas elípticas *Elliptic Curve-Diffie-Hellman*
EC-DHE	Diffie-Hellman basado en curvas elípticas con clave efímera
EC-DLOG	Logaritmo discreto sobre curvas elípticas *Elliptic Curve Discrete Logarithm*
EC-GDSA	Algoritmo de firma digital alemán *Elliptic Curve German Digital Signature Algorithm*
EC-KCDSA	Algoritmo de firma digital coreano basado en certificados *Elliptic Curve Korean Certificate-based Digital Signature Algorithm*
EC-Schnorr	Algoritmo de firma de Schnorr para curvas elípticas *Elliptic Curve Based Schnorr Signature Algorithm*
ECDLP	Problema del logaritmo discreto sobre curvas elípticas *Elliptic Curve Discrete Logarithm Problem*
ECDSA	Algoritmo de firma digital basado en curvas elípticas *Elliptic Curve Digital Signature Algorithm*
ECIES	Esquema de cifrado integrado basado en curvas elípticas *Elliptic Curve Integrated Encryption Scheme*
ECRYPT	*European Network of Excellence in Cryptology*
ECP	Curva elípticas modulo un primo *Elliptic Curve modulo a Prime*
ESP	Carga útil de seguridad encapsulada *Encapsulated Security Payload*
ESSIV	*Encrypted Salt-Sector IV*
EUF-CMA	Existencialmente infalsificable bajo un ataque adaptativo al mensaje elegido

	Existentially UnForgeable under adaptive Chosen Message Attack
Falcon	*FAst-Fourier Lattice-based COmpact signatures over NTRU*
FE2OS	*Field Element to Octet String Conversion Primitive*
FFDHE	Mecanismo de intercambio de claves efímeras Diffie-Hellman basado en cuerpos finitos *Finite-Field-based Diffie-Hellman Ephemeral key exchange mechanism*
FF-DLOG	Logaritmo discreto multiplicativo sobre un cuerpo finito *Finite Field-Discrete Logarithm*
FTS	*Few-Time Signature*
FTPS	Protocolo seguro de transferencia de archivos *File Transfer Protocol Secure*
FORS	Bosque de subconjuntos aleatorios *Forest Of Random Subsets*
FW	Firmware
GCM	Contador de Galois *Galois Counter Mode*
GMAC	MAC basado en el modo contador de Galois *Galois Message Authentication Code*
HBS	Firma basada en *hashes* *Hash-Based Signature*
HMAC	MAC basado en función resumen *Hash function-based MAC*
HQC	*Hamming Quasi-Cyclic*
HTTPS	Protocolo seguro de transferencia de hipertexto *Hypertext Transfer Protocol Secure*
HSM	Módulo de seguridad hardware *Hardware Security Module*
I2SO	*Integer to Octet String Conversion Primitive*
IFP	Problema de la factorización de números enteros *Integer Factorization Problem*
IND-CCA1	Indistinguibilidad por ataque al texto cifrado elegido *INDistinguishability under Chosen Ciphertext Attack*
IND-CCA2	Indistinguibilidad por ataque adaptativo al texto cifrado elegido *INDistinguishability under Adaptive Chosen Ciphertext Attack*
IND-CPA	Indistinguibilidad por ataque al texto claro elegido *INDistinguishability under Chosen-Plaintext Attack*
IKE	Intercambio de claves de Internet *Internet Key Exchange*
IKEv2	Versión 2 del protocolo IKE
IP	Protocolo de Internet *Internet Protocol*
IPsec	Seguridad del protocolo de Internet *Internet Protocol Security*

IV	Vector de inicialización *Initialization Vector*
KA	Acuerdo de clave *Key Agreement*
KCDSA	Algoritmo de firma digital coreana basado en certificados *Korean Certificate-based Digital Signature Algorithm*
KDF	Función de derivación de claves *Key Derivation Function*
KEM	Mecanismo de encapsulamiento de claves *Key Encapsulation Mechanism*
KMAC	*Keccak Message Authentication Code*
KW	*Key Wrap*
KWP	*Key Wrap with Padding*
LWE	Aprendizaje con errores *Learning With Errors*
MAC	Código de autenticación de mensaje *Message Authentication Code*
MitM	Hombre en el medio *Man-in-the-Middle* *Man-in-the-Middle*
ML-DSA	*Module-Lattice-Based Digital Signature Standard*
ML-KEM	*Module-Lattice-Based Key-Encapsulation Mechanism Standard*
MLWE	Aprendizaje con errores sobre módulos *Module Learning With Errors*
MSS	Esquema de firma de Merkle *Merkle Signature Scheme*
MODP	Exponenciación modular *Modular exponentiation*
NIST	*National Institute for Standards and Technology*
NPTRNG	Generador no físico de números realmente aleatorios *Non-Physical True Random Number Generator*
NTRU	*N-th degree Truncated polynomial Ring Units*
NTT	Transformada teórica de números *Number Theoretic Transform*
OAEP	Relleno de cifrado asimétrico óptimo *Optimal Asymmetric Encryption Padding*
OFB	Realimentación de la salida *Output Feedback*
OID	Identificador del objeto *Object Identifier*
OS2I	*Octet String to Integer Conversion*
OTS	Firma única *One Time Signature*
OWA	Ataque unidireccional

	One-Way Attack
PFS	Secreto perfecto persistente *Perfect Forward Secrecy*
PIN	Número de identificación personal *Personal Identification Number*
PKE	Sistema de clave pública *Public Key Encryption*
PKCS	*Public-Key Cryptography Standard*
PPKE	Cifrado de clave pública probabilístico *Probabilistic Public Key Encryption*
PQ	Postcuántico (*Post-Quantum*
PQC	Criptografía postcuántica *Post-Quantum Cryptography*
PRF	Función pseudo-aleatoria *Pseudo-Random Function*
PSK	Claves precompartidas *Pre-Shared Keys*
PSS	Firma digital probabilística *Probabilistic Signature Scheme*
PTRNG	Generador físico de números realmente aleatorios *Physical True Random Number Generator*
PSK	Clave precompartida *Pre-Shared Key*
QCSD	*Quasi-Cyclic Syndrome Decoding*
QROM	Modelo del oráculo aleatorio cuántico *Quantum Random Oracle Model*
RBG	Generador de bits aleatorios *Random Bit Generator*
RFID	Etiquetas de identificación por radio frecuencia *Radio Frequency Identification*
RLWE	Aprendizaje con errores sobre anillos *Ring Learning With Errors*
RNG	Generador de números aleatorios *Random Number Generator*
ROM	Modelo del oráculo aleatorio *Random Oracle Model*
RSA	Rivest, Shamir y Adleman
SA	Asociación de seguridad *Security Association*
SEC	*Standards for Efficient Cryptography*
SHAKE	*Secure Hash Algorithm and Keccak*
SIDH	Intercambio de clave mediante isogenias tipo Diffie-Hellman *Supersingular Isogeny Diffie-Hellman key exchange*

SIKE	*Supersingular Isogeny Key Encapsulation*
SIS	Problema de la solución entera más corta *Short Integer Solution*
SIV	Vector de inicialización sintético *Synthetic Initialization Vector*
SKE	Sistema de clave secreta (*Secret Key Encryption*)
SLH-DSA	*Stateless Hash-Based Digital Signature Standard*
SOG-IS	*Senior Officials Group-Information Systems Security*
SSCDHP	Problema de Diffie-Hellman computacional supersingular *Supersingular Computational Diffie-Hellamn Problem*
SSIP	Problema de isogenias supersingulares *Supersingular Isogeny Problem*
SSH	*Secure Shell*
SSL	Capa de conexión segura *Secure Socket Layer*
SVP	Problema del vector más corto *Shortest Vector Problem*
SW	Software
TCP/IP	*Transmission Control Protocol/Internet Protocol*
TLS	Seguridad de la capa de transporte *Transport Layer Security*
TRNG	Generador de números realmente aleatorios *True Random Number Generator*
UOV	Aceite y vinagre desequilibrado *Unbalanced Oil and Vinegar*
VPN	Red privada virtual *Virtual Private Network*
WOTS$^+$	*Winternitz One-Time Signature Plus*
XEX	*XOR Encrypt XOR*
XMSS	Esquema extendido de firma de Merkle *eXtended Merkle Signature Scheme*
XOF	Función con salida extensible *Extendable-Output Function*
XTS	Cifrado en bloque modificable XEX con robo de texto cifrado *XEX Tweakable Block Cipher with Ciphertext Stealing*

Referencias

[ABD+15] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelink, and P. Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *Proc. 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS'15)*, pages 5–17, 2015. http://dx.doi.org/10.1145/2810103.2813707. 63

[ABD+23] E. Alkim, J. W. Bos, L. Ducas, P. Longa, I. Mironov, M. Naehrig, V. Nikolaenko, C. Peikert, A. Raghunathan, and D. Stebila. FrodoKEM: Learning with errors key encapsulation. preliminary standardization proposal. Online publication, 2023. https://frodokem.org/files/FrodoKEM-standard_proposal-20230314.pdf. 158

[ABP+13] N. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. C. Schuldt. On the security of RC4 in TLS and WPA. In *Proc. 22nd USENIX Security Symposium*, pages 305–320, 2013. https://www.usenix.org/conference/usenixsecurity13/technical-sessions/paper/alFardan. 64

[ACM1.3] SOG-IS. *SOG-IS Crypto Evaluation Scheme. Agreed Cryptographic Mechanisms. Version 1.3.* Senior Officials Group-Information Systems Security, Crypto Working Group, 2023. https://www.sogis.eu/uk/supporting_doc_en.html. 9, 13, 80, 102

[AKS04] M. Agrawal, N. Kayal, and N. Saxena. Primes is in P. *Ann. of Math.*, 160(2):781–793, 2004. https://doi.org/10.4007/annals.2004.160.781. 100

[ANS11] ANSSI. *Avis relatif aux paramètres de courbes ellitpiques définis pas l'État français*. Agence Nationale de la Sécurité des Systèmes d'Information, Journal Officiel 0241 (Oct. 2011), p. 17533, 2011. https://www.legifrance.gouv.fr/jorf/id/JORFTEXT000024668816. 47

[ANS20a] ANSSI. *Guide des Mécanismes Cryptographiques, Règles et Recommandations Concernant le Choix et le Dimensionnement des Mécanismes Cryptographiques*. Agence Nationale de la Sécurité des Systèmes d'Information, PG-083, 1/1/2020,

2020. https://www.ssi.gouv.fr/uploads/2021/03/anssi-guide-mecanismes_crypto-2.04.pdf. 9, 80

[ANS20b] ANSSI. *Recommandations de sécurité relatives à TLS*. Agence Nationale de la Sécurité des Systèmes d'Information, 2020. https://www.ssi.gouv.fr/guide/recommandations-de-securite-relatives-a-tls/. 60

[ANS20c] ANSSI. *Recommandations pour les Architectures des Systèmes d'Information Sensibles ou Diffusion Restreinte. Version 1.1*. Agence Nationale de la Sécurité des Systèmes d'Information, 2020. https://www.ssi.gouv.fr/guide/recommandations-pour-les-architectures-des-systemes\-dinformation-sensibles-ou-diffusion-restreinte/. 9

[ANS21] ANSSI. *Guide de Sélection D'Algorithmes Cryptographiques, v1.0*. Agence Nationale de la Sécurité des Systèmes d'Information, PA-079, 8/3/2021, 2021. https://www.ssi.gouv.fr/uploads/2021/03/anssi-guide-selection_crypto-1.0.pdf. 9, 80

[ANSIX9.6] ANSI. *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*. American National Standards Institute, ANSI X9.62:2005, 2005. https://standards.globalspec.com/std/1955141/ANSI20X9.62. 166

[ANSIX9.63] ANSI. *Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography (R2017)*. American National Standards Institute, ANSI X9.63:2011, 2017. https://webstore.ansi.org/standards/ascx9/ansix9632011r2017. 37, 38

[Ant22] S. Antonov. Round 3 official comment: SPHINCS+. Online publication, 2022. https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/FVItvyRea28/m/mGaRi5iZBwAJ. 142

[AS15] J. Alwen and V. Serbinenko. High parallel complexity graphs and memory-hard functions. In *Proc. ACM symposium on Theory of Computing (STOC'15)*, pages 595–603, 2015. https://doi.org/10.1145/2746539.2746622. 39

[BBJ+08] D. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters. Twisted Edwards curves. *Cryptology ePrint Archive, Report 2008/013*, 2008. https://eprint.iacr.org/2008/013. 169

[BBK03] E. Barkan, E. Biham, and N. Keller. Instant ciphertext-only cryptanalysis of GSM encrypted communication. In *Proc. Annual International Cryptology Conference, Advances in Cryptology - CRYPTO 2003, Lecture Notes Comput. Sci.*, volume 2729, pages 600–616, 2003. https://doi.org/10.1007/978-3-540-45146-4_35. 17

[BCD+06] J. Buchmann, L. C. Coronado García, E. Dahmen, M. Döring, and E. Klintsevich. CMSS - an improved Merkle signature scheme. In *Progress in Cryptology - INDOCRYPT 2006, Lecture Notes Comput. Sci.*, volume 4329, pages 349–363, 2006. https://doi.org/10.1007/11941378_25. 156

[BCD+16] J. Bos, C. Costello, L. Ducas, I. Mironov, M. Naehrig, V. Nikolaenko, A. Raghunathan, and D. Stebila. Frodo: Take off the ring! Practical, quantum-secure key exchange from LWE. In *Proc. 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS'16*, pages 1006–1018, 2016. https://doi.org/10.1145/2976749.2978425. 126

[BDE+11] J. Buchmann, E. Dahmen, S. Ereth, A. Hülsing, and M. Rückert. On the security of the Winternitz one-time signature scheme. In *Proc. Annual International Cryptology Conference in Africa, Progress in Cryptology - AFRICACRYPT 2011, Lecture Notes Comput. Sci.*, volume 6737, pages 363–378, 2011. https://doi.org/10.1007/978-3-642-21969-6_23. 156

[BDH11] J. Buchmann, E. Dahmen, and A. Hülsing. XMSS - a practical forward secure signature scheme based on minimal security assumptions. In *Proc. Post-Quantum Cryptography (PQCrypto 2011), Lecture Notes Comput. Sci.*, volume 7071, pages 117–129, 2011. https://doi.org/10.1007/978-3-642-25405-5_8. 54, 154, 156

[BDK+07] J. Buchmann, E. Dahmen, E. Klintsevich, K. Okeya, and C. Vuillaume. Merkle signatures with virtually unlimited signature capacity. In *Proc. International Conference on Applied Cryptography and Network Security (ACNS 2007), Lecture Notes Comput. Sci.*, volume 4521, pages 31–45, 2007. https://doi.org/10.1007/978-3-540-72738-5_3. 156

[BDK16] A. Biryukov, D. Dinu, and D. Khovratovich. Argon2: New generation of memory-hard functions for password hashing and other applications. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 292–302, 2016. https://doi.org/10.1109/EuroSP.2016.31. 39

[BDS08] J. Buchmann, E. Dahmen, and M. Schneider. Merkle tree traversal revisited. In *Proc. Post-Quantum Cryptography (PQCrypto 2016), Lecture Notes Comput. Sci.*, volume 5299, pages 63–78, 2008. https://doi.org/10.1007/978-3-540-88403-3_5. 156

[BDS09] J. Buchmann, E. Dahmen, and M. Szydlo. *Post-Quantum Cryptography*, chapter Hash-based Digital Signature Schemes, pages 35–93. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. https://doi.org/10.1007/978-3-540-88702-7_3. 156

[Ber05] D. Bernstein. The Poly1305-AES message-authentication code. In *Proc. International Workshop Fast Software Encryption, FSE'2005, Lecture Notes Comput. Sci.*, 2005. 35

[Ber08] D. J. Bernstein. ChaCha, a variant of Salsa20. In *Proc. Workshop Record of SASC 2008: The State of the Art in Stream Ciphers*, pages 1–6, 2008. https://cr.yp.to/papers.html#chacha. 35

[BHH+15] D. J. Bernstein, D. Hopwood, A. Hülsing, T. Lange, R. Niederhagen, L. Papachristodoulou, M. Schneider, P. Schwabe, and Z. Wilcox-O'Hearn. SPHINCS: practical stateless hash-based signatures. In *Proc. Annual International Cryptology Conference, Advances in Cryptology - EUROCRYPT 2015, Lecture Notes Comput. Sci.*, volume 9056, pages 368–397, 2015. https://doi.org/10.1007/978-3-662-46800-5_15. 156

[BL07] D. J. Bernstein and T. Lange. Faster addition and doubling on elliptic curves. *Lecture Notes Comput. Sci.*, 4833:29–50, 2007. https://doi.org/10.1007/978-3-540-76900-2_3. 169

[Ble98] D. Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In *Proc. Annual International Cryptology Conference, Advances in Cryptology - CRYPTO 1998, Lecture Notes Comput. Sci.*, volume 14162, pages 1–12, 1998. https://doi.org/10.1007/BFb0055716. 17, 53

[BN00] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *Proc. International Conference on the Theory and Application of Cryptology and Information Security, Advances in Cryptology - ASIACRYPT 2000, Lecture Notes Comput. Sci.*,

volume 1976, pages 531–545, 2000. https://doi.org/10.1007/3-540-36178-2_1. 35

[BR95] M. Bellare and P. Rogaway. Optimal asymmetric encryption - How to encrypt with RSA. In *Proc. Annual International Cryptology Conference, Advances in Cryptology - EUROCRYPT 1994, Lecture Notes Comput. Sci.*, volume 950, pages 92–111, 1995. https://doi.org/10.1007/BFb0053428F. 50

[BSI13a] BSI. *Functionality Classes and Evaluation Methodology for Deterministic Random Number Generators (AIS-20)*. Bundesamt für Sicherheit in der Informationstechnik, version 3, 2013. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_20_pdf.pdf. 80, 83

[BSI13b] BSI. *Functionality Classes and Evaluation Methodology for Physical Random Number Generators (AIS-31)*. Bundesamt für Sicherheit in der Informationstechnik, version 3, 2013. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_31_pdf.pdf. 80, 82, 83

[BSI22] BSI. *Cryptographic Mechanisms: Recommendations and Key Lengths, version 2022-01*. Bundesamt für Sicherheit in der Informationstechnik, BSI TR-02102-1, 2022/01/28, 2022. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.pdf. 9, 80, 81, 85, 87

[CCN22] CCN. *Recomendaciones para una transición postcuántica segura, CCN-TEC 009*. Centro Criptológico Nacional, Ministerio de Defensa, España, 2022. https://www.ccn.cni.es/index.php/es/docman/documentos-publicos/boletines-pytec/495-ccn-tec-009-recomendaciones-transicion-postcuantica-segura/file. 157

[CD22] W. Castryck and T. Decru. An efficient key recovery attack on SIDH (preliminary version). *Cryptology ePrint Archive, Report 2022/975*, 2022. https://eprint.iacr.org/2022/975. 109

[CDF+21] C. Cremers, S. Düzlü, R. Fiedler, C. Janson, and M. Fischlin. BUFFing signature schemes beyond unforgeability and the case of post-quantum signatures. In *Proc. IEEE Symposium on Security and Privacy, SP'2021*, pages 1696–1714, 2021. https://doi.org/10.1109/SP40001.2021.00093. 127

[CLM20] L. Chen, T. Lange, and S. Moriai (Eds.). *WG 2 SD8 (Post-Quantum Cryptography) – Part 1: General*. International Organization for Standardization, 2020. https://isotc.iso.org/ecom/livelink/open/jtc1sc27wg2. 106

[DGH16] R. Durán Díaz, V. Gayoso Martínez, and L. Hernández Encinas. Generación de primos demostrables: implementación y resultados. In *Proc. XIV Reunión Española de Criptología y Seguridad de la Información (RECSI 2016)*, pages 58–63, 2016. 99

[DH76] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, 22:644–654, 1976. htpps://doi.org/10.1109/TIT.1976.1055638. 56

[DHM05] R. Durán Díaz, L. Hernández Encinas, and J. Muñoz Masqué. *El criptosistema RSA*. RA-MA, Madrid, España, 2005. https://www.ra-ma.es/libro/el-criptosistemarsa_48854/. 100

[DLP93] I. Damgård, P. Landrock, and C. Pomerance. Average case error estimates for the strong provable prime test. *Mathematics of Computation*, 61(203):177–194, 1993. https://doi.org/10.2307/2152945. 101

[DSS05] C. Dods, N. Smart, and M. Stam. Hash based digital signature schemes. In *Proc. IMA International Conference on Cryptography and Coding, (IMACC 2005), Lecture Notes Comput. Sci.*, volume 3796, pages 96–115, 2005. https://doi.org/10.1007/11586821_8. 156

[Edw07] H. Edwards. A normal form for elliptic curves. *Buyll. Amer. Math. Soc.*, 44(3):393–422, 2007. https://doi.org/10.1090/S0273-0979-07-01153-6. 169

[EH95] T. ElGamal and K. E. Hickman. *The SSL Protocol*. Internet Engineering Task Force, 1995. https://datatracker.ietf.org/doc/html/draft-hickman-netscape-ssl-00. 61

[ETS20] ETSI. ETSI TS 103 744 v1.1.1 (2020-12). CYBER. Quantum-safe hybrid key exchanges. Technical report, 2020. https://www.etsi.org/deliver/etsi_ts/103700_103799/103744/01.01.01_60/ts_103744v010101p.pdf. 40, 161, 162, 163

[FIPS-140IG] NIST. *Implementation Guidance for FIPS 140-3 and the Cryptographic Module Validation Program. CMVP*. National Institute of Standard and Technology, FIPS-140 IG,

2024. https://csrc.nist.gov/csrc/media/Projects/cryptographic-module-validation-program/documents/fips140-3/FIPS140-3IG.pdf. 30

[FIPS140-3] NIST. *Security Requirements for Cryptographic Modules*. National Institute of Standard and Technology, Federal Information Processing Standard Publication, FIPS PUB 140-3, 2019. https://doi.org/10.6028/NIST.FIPS.140-3. 138

[FIPS180-4] NIST. *Secure Hash Standard (SHS)*. National Institute of Standard and Technology, NIST FIPS 180-4, 2015. https://doi.org/10.6028/NIST.FIPS.180-4. 25, 31, 136

[FIPS186-4] NIST. *Digital Signature Standard (DSS)*. National Institute of Standard and Technology, NIST FIPS 186-4, 2013. https://doi.org/10.6028/NIST.FIPS.186-4. 44, 52, 165, 166

[FIPS186-5] NIST. *Digital Signature Standard (DSS)*. National Institute of Standard and Technology, NIST FIPS 186-5, 2023. https://doi.org/10.6028/NIST.FIPS.186-5. 52, 101, 107, 169, 170, 173, 174

[FIPS197] NIST. *Advanced Encryption Standard*. National Institute of Standard and Technology, Federal Information Processing Standard Publication, FIPS 197, 2023. https://doi.org/10.6028/NIST.FIPS.197-upd1. 23

[FIPS202] NIST. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. National Institute of Standard and Technology, NIST FIPS 202, 2015. https://doi.org/10.6028/NIST.FIPS.202. 25, 26, 129, 136, 174

[FIPS203] NIST. *Module-Lattice-Based Key-Encapsulation Mechanism Standard*. National Institute of Standard and Technology, NIST FIPS 203, 2024. https://doi.org/10.6028/NIST.FIPS.203. 57, 58, 114, 115, 116, 117, 125, 127, 158

[FIPS204] NIST. *Module-Lattice-Based Digital Signature Standard*. National Institute of Standard and Technology, NIST FIPS 204, 2024. https://doi.org/10.6028/NIST.FIPS.204. 52, 54, 55, 127, 129, 139, 158

[FIPS205] NIST. *Stateless Hash-Based Digital Signature Standard*. National Institute of Standard and Technology, NIST FIPS 205, 2024. https://doi.org/10.6028/NIST.FIPS.205. 52, 54, 55, 141, 142, 144, 158

[FKK11] A. O. Freier, P. Karlton, and P. C. Kocher. *The Secure Sockets Layer (SSL) Protocol, Version 3.0*. Internet Engineering Task Force, 2011. https://datatracker.ietf.org/doc/html/draft-hickman-netscape-ssl-00. 61

[FRODO-KEM] E. Alkim, J. W. Bos, L. Ducas, P. Longa, I. Mironov, M. Naehrig, V. Nikolaenko, C. Peikert, A. Raghunathan, and D. Stebila. FrodoKEM learning with errors key encapsulation (Round 3 Submission). Online publication, 2021. https://frodokem.org/. 57, 58, 126

[GHM18] V. Gayoso Martínez, L. Hernández Encinas, and A. Martín Muñoz. *Criptografía con Curvas Elípticas*. Consejo Superior de Investigaciones Científicas (CSIC), Madrid, Spain, 1^a edition, 2018. http://editorial.csic.es/publicaciones/libros/13133/0/criptografia-con-curvas-elipticas.html. 45

[Gro96] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proc. 28th annual ACM symposium on Theory of Computing (STOC'96)*, pages 212–219, 1996. https://doi.org/10.1145/800070.802214. 107

[Gro97] L. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Phys. Rev. Lett.*, 79(2):325–328, 1997. https://doi.org/10.1103/PhysRevLett.79.325. 12, 107

[HBD+20] A. Hulsing, D. J. Bernstein, C. Dobraunig, M. Eichlseder, S. Fluhrer, S.-L. Gazdag, P. Kampanakis, S. Kolbl, T. Lange, M. M. Lauridsen, F. Mendel, R. Niederhagen, C. Rechberger, J. Rijneveld, P. Schwabe, and J.-P. Aumasson. SPHINCS+. Online publication, 2020. https://sphincs.org/. 55, 151

[HMQ03] L. Hernández Encinas, J. Muñoz Masqué, and A. Queiruga Dios. Large decryption exponents in RSA. *Applied Mathematics Letters*, 16(3):292–295, 2003. https://doi.org/10.1016/S0893-9659(03)80046-0. 104

[HMV04] D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag, New York, NY, USA, 2004. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.394.3037&rep=rep1&type=pdf. 46

[Hül13] A. Hülsing. W-OTS+ – shorter signatures for hash-based signature schemes. In *Proc. International Conference on Cryptology in Africa – AFRICACRYPT 2013, Lecture Notes Comput. Sci.*, volume 7918, pages 173–188, Berlin, Heidelberg,

2013. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-38553-7_10. 156

[IEEE1363a] IEEE 1363a. *Standard Specifications for Public Key Cryptography-Amendment 1: Additional Techniques*. Institute of Electrical and Electronics Engineers, IEEE 1363a, 2004. https://ieeexplore.ieee.org/document/1335427. 166

[ISO10116] ISO/IEC 10116. *Information technology – Security techniques – Modes of operation for an n-bit block cipher*. International Organization for Standardization/International Electrotechnical Commission, 2017. https://www.iso.org/standard/64575.html. 28

[ISO10118-3] ISO/IEC 10118-3. *ISO/IEC 10118-3:2018 - Information technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions. ISO/IEC10118-3:2018*. International Organization for Standardization/International Electrotechnical Commission, 2018. https://www.iso.org/standard/67116.html. 25

[ISO11770-3] ISO/IEC 11770-3. *Information technology – Security techniques – Key management, Part 3: Mechanisms using asymmetric techniques, ISO/IEC 11770-3*. International Organization for Standardization, 2015. https://www.iso.org/standard/60237.html. 57

[ISO14888-3] ISO/IEC 14888-3. *Information technology – Security techniques – Digital signatures with appendix, Part 3: Discrete logarithm based mechanisms, ISO/IEC 14888-3*. International Organization for Standardization, 2018. https://www.iso.org/standard/76382.html. 52, 166

[ISO18031] ISO/IEC 18031. *Information Technology – Security Techniques – Random bit generation, ISO/IEC 18031*. International Organization for Standardization/International Electrotechnical Commission, 2011. https://www.iso.org/standard/54945.html. 84

[ISO18033-2] ISO/IEC 18033-2. *Information Technology-Security Techniques-Encryption Algorithms-Part 2: Asymmetric Ciphers*. International Organization for Standardization/International Electrotechnical Commission, 2006. https://www.iso.org/standard/37971.html. 57, 58

[ISO18033-3] ISO/IEC 18033-3. *Information Technology – Security Techniques – Encryption Algorithms – Part 3: Block Ciphers*. International

Organization for Standardization/International Electrotechnical Commission, 2010. https://www.iso.org/standard/54531.html. 23

[ISO19772] ISO/IEC 19772. *Information technology – Authenticated encryption*. International Organization for Standardization/International Electrotechnical Commission, 2020. https://www.iso.org/standard/81550.html. 35

[ISO9796-2] ISO/IEC 9796-2. *Information technology – Security techniques – Digital signature schemes giving message recovery, Part 2: Integer factorization based mechanisms, ISO/IEC 9796-2*. International Organization for Standardization, 2010. https://www.iso.org/standard/54788.html. 52

[ISO9797-1] ISO/IEC 9797-1. *Information technology – Security techniques – Message Authentication Codes (MACs), Part 1: Mechanisms using a block cipher, ISO/IEC 9797-1*. International Organization for Standardization/International Electrotechnical Commission, 2011. https://www.iso.org/standard/50375.html. 31

[ISO9797-2] ISO/IEC 9797-2. *Information technology – Security techniques – Message Authentication Codes (MACs), Part 2: Mechanisms using a dedicated hash-function*. International Organization for Standardization/International Electrotechnical Commission, 2011. https://www.iso.org/standard/51618.html. 31

[JZC+17] H. Jiang, Z. Zhang, L. Chen, H. Wang, and Z. Ma. IND-CCA-secure key encapsulation mechanism in the quantum random oracle model, revisited. *Cryptology ePrint Archive, Report 2017-1096*, 2017. http://eprint.iacr.org/2017/1096. 126

[KCD98] KCDSA Task Force Team. *The Korean Certificate-based Digital Signature Algorithm*. International Organization for Standardization, 1998. https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.99.8398. 177

[KLLNP16a] M. Kaplan, G. Leurent, A. Leverrier, and M. Naya-Plasencia. Breaking symmetric cryptosystems using quantum period finding. In *Proc. Annual International Cryptology Conference, Advances in Cryptology - CRYPTO 2016, Lecture Notes Comput. Sci.*, volume 9815, pages 207–237, 2016. https://doi.org/10.1007/978-3-662-53008-5_. 12

[KLLNP16b] M. Kaplan, G. Leurent, A. Leverrier, and M. Naya-Plasencia. Quantum differential and linear cryptanalysis. *IACR Transactions*

on Symmetric Cryptology, 2:71–94, 2016. https://doi.org/10.13154/tosc.v2016.i1.71-94. 12

[KS11] W. Killmann and W. Schindler. *Functionality Classes and Evaluation Methodology for Physical Random Number Generators*. Bundesamt für Sicherheit in der Informationstechnik, version 2, 2011. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_31_Functionality_classes_for_random_number_generators_e.pdf. 80, 82, 83, 85, 86

[Lam79] L. Lamport. Constructing digital signatures from a one way function. Technical Report Technical Report SRI-CSL-98, SRI International, Computer Science Laboratory, 1979. https://www.microsoft.com/en-us/research/publication/constructing-digital-signatures-one-way-function/. 154, 155

[LDK+20] V. Lyubashevsky, L. Ducas, E. Kiltz, T. Lepoint, P. Schwabe, G. Seiler, and D. Stehle. CRYSTALS-DILITHIUM. Online publication, 2020. https://pq-crystals.org/dilithium/index.shtml. 55

[LL99] C. H. Lim and P. J. Lee. The Korean certificate-based digital signature algorithm. *Computers and Electrical Engineering*, 25:249–265, 1999. https://doi.org/10.1016/S0045-7906(99)00011-7. 177

[Mat94] M. Matsui. Linear cryptanalysis method for DES cipher. *Lecture Notes Comput. Sci.*, 765:386–397, 1994. https://doi.org/10.1007/3-540-48285-7_33. 17

[Mau95] U. Maurer. Fast generation of prime numbers and secure publickey cryptographic parameters. *Journal of Cryptology*, 8(3):123–155, 1995. https://doi.org/10.1007/BF00202269. 99

[Mer89] R. C. Merkle. A certified digital signature. In *Proc. Annual International Cryptology Conference, Advances in Cryptology - CRYPTO 1989, Lecture Notes Comput. Sci.*, volume 435, pages 218–238, 1989. https://doi.org/10.1007/0-387-34805-0_21. 54, 154, 156

[Mil76] G. L. Miller. Riemann's hypothesis and test for primality. *J. Comput. & System Sci.*, 13(3):300–317, 1976. https://doi.org/10.1016/S0022-0000(76)80043-8. 100

[MvV96] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1996. http://cacr.uwaterloo.ca/hac/. 24, 100

[Nem16] M. Nemec. The properties of RSA key generation process in software libraries. Technical report, Master Thesis, Faculty odf Informatics, Masaryk University, Brno, 2016. http://www.fi.muni.cz/reports. 104

[NIS17] NIST. Post-quantum cryptography. On-line publication, 2017. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography. 48, 107

[NIS20] NIST. PQC standardization process: Third round candidate announcement. Online publication, 2020. https://csrc.nist.gov/News/2020/pqc-third-round-candidate-announcement. 58

[NIS22] NIST. Post-quantum cryptography. selected algorithms 2022. On-line publication, 2022. https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022. 55, 58, 108

[NK09] K. Nohl and S. Krißler. Subverting the security base of GSM. Technical report, 2009. https://web.archive.org/web/20110726142029/https://har2009.org/program/attachments/119_GSM.A51.Cracking.Nohl.pdf. 17

[NSv+17] M. Nemec, M. Sys, P. Švenda, D. Klinec, and V. Matyas. The return of Coppersmith's attack: Practical factorization of widely used RSA moduli. In *Proc. 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS'17)*, pages 496–499, 2017. https://doi.org/10.1145/3133956.3133969. 99, 104

[Per09] C. Percival. Stronger key derivation via sequential memory-hard functions, 2009. http://www.tarsnap.com/scrypt/scrypt.pdf. 39

[PKC22] R. Perlner, J. Kelsey, and D. Cooper. Breaking category five SPHINCS+ with SHA-256. In *Proc. Post-Quantum Cryptography (PQCrypto 2022), Lecture Notes Comput. Sci.*, volume 13512, pages 501–522, 2022. https://doi.org/10.1007/978-3-031-17234-2_23. 142

[Rab80] M. Rabin. Probabilistic algorithms for testing primality. *J. Number Theory*, 12(1):128–138, 1980. https://doi.org/10.1016/0022-314X(80)90084-0. 100

[RD421/2004] Ministerio de Defensa. *Real Decreto 421/2004, de 12 de marzo, por el que se regula el Centro Criptológico Nacional*. BOE núm. 68, de 19 de marzo de 2004, páginas 12203 a 12204, 2004. https://www.boe.es/eli/es/rd/2004/03/12/421. 9

[Reg04] O. Regev. New lattice-based cryptographic constructions. *Journal of the ACM*, 51(6):899–942, 2004. https://doi.org/10.1145/1039488.1039490. 113

[RFC2104] H. Krawczyk, M. Bellare, and R. Canetti. *HMAC: Keyed-Hashing for Message Authentication*. Internet Engineering Task Force, RFC 2104, 1997. https://tools.ietf.org/html/rfc2104. 31

[RFC2246] T. Dierks and C. Allen. *The TLS Protocol, Version 1.0*. Internet Engineering Task Force, RFC 2246, 1999. https://tools.ietf.org/html/rfc2246. 61

[RFC2898] B. Kaliski. *PKCS #5: Password-Based Cryptography Specification, Version 2.0*. Internet Engineering Task Force, RFC 2898, 2000. https://datatracker.ietf.org/doc/html/rfc2898. 38

[RFC3526] T. Kivinen and M. Kojo. *More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)*. Network Working Group, 2003. https://datatracker.ietf.org/doc/html/rfc3526. 44, 72, 76, 77

[RFC4055] J. Schaad, B. Kaliski, and R. Housley. *Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. Internet Engineering Task Force, RFC 4055, 2005. https://tools.ietf.org/html/rfc4055. 79

[RFC4251] T. Ylönen and C. Lonvick. *The Secure Shell (SSH) Protocol Architecture*. Internet Engineering Task Force, RFC 4251, 2006. https://tools.ietf.org/html/rfc4251. 71, 72

[RFC4252] T. Ylönen and C. Lonvick. *The Secure Shell (SSH) Authentication Protocol*. Internet Engineering Task Force, RFC 4252, 2006. https://tools.ietf.org/html/rfc4252. 72

[RFC4253] T. Ylönen and C. Lonvick. *The Secure Shell (SSH) Transport Layer Protocol*. Internet Engineering Task Force, RFC 4253, 2006. https://tools.ietf.org/html/rfc4253. 72, 73, 74

[RFC4254] T. Ylönen and C. Lonvick. *The Secure Shell (SSH) Connection Protocol*. Internet Engineering Task Force, RFC 4254, 2006. https://tools.ietf.org/html/rfc4254. 72

[RFC4303] S. Kent. *IP Encapsulating Security Payload (ESP)*. Internet Engineering Task Force, RFC 4303, 2005. https://tools.ietf.org/html/rfc4303. 76

[RFC4344] M. Bellare, T.Kohno, and C. Namprempre. *The Secure Shell (SSH) Transport Layer Encryption Modes*. Internet Engineering Task Force, RFC 4344, 2006. https://tools.ietf.org/html/rfc4344. 74

[RFC4346] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol, Version 1.1*. Internet Engineering Task Force, RFC 4346, 2006. https://tools.ietf.org/html/rfc4346. 61

[RFC4419] M. Friedl, N. Provos, and W. Simpon. *Diffie-Hellman Group Exchange for the Secure Shell (SSH) Transport Layer Protocol*. Internet Engineering Task Force, RFC 4419, 2006. https://tools.ietf.org/html/rfc4419. 72

[RFC4494] J. Song and J. Lee. *The AES-CMAC-96 Algorithm and Its Use with IPsec*. Internet Engineering Task Force, RFC 4494, 2006. https://tools.ietf.org/html/rfc4494. 78

[RFC4543] A. Conta, S. Deering, and M. Gupta. *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*. Internet Engineering Task Force, RFC 4543, 2006. https://tools.ietf.org/html/rfc4543. 78

[RFC4615] J. Song, R. Poovendran, J. Lee, and T. Iwata. *The Advanced Encryption Standard-Cipher-based Message Authentication Code-Pseudo-Random Function-128 (AES-CMAC-PRF-128) Algorithm for the Internet Key Exchange Protocol (IKE)*. Internet Engineering Task Force, RFC 4615, 2006. https://tools.ietf.org/html/rfc4615. 78

[RFC4754] D. Fu and J. Solinas. *IKE and IKEv2 Authentication Using the Elliptic Curve Digital Signature Algorithm (ECDSA)*. Internet Engineering Task Force, RFC 4754, 2007. https://tools.ietf.org/html/rfc4754. 79

[RFC4868] S. Kelly and S. Frankel. *Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec*. Internet Engineering Task Force, RFC 4868, 2007. https://tools.ietf.org/html/rfc4868. 78

[RFC5246] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol, Version 1.2*. Internet Engineering Task Force, RFC 5246, 2008. https://tools.ietf.org/html/rfc5246. 60, 61, 62, 69, 71

[RFC5288] J. Salowey, D. McGrew, and A. Choudhury. *AES Galois Counter Mode (GCM) Cipher Suites for TLS*. Internet Engineering Task Force, RFC 5288, 2008. https://tools.ietf.org/html/rfc5288. 69

[RFC5289] E. Rescorla. *TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)*. Internet Engineering Task Force, RFC 5289, 2008. https://tools.ietf.org/html/rfc5289. 68, 69

[RFC5297] D. Harkins. *Synthetic Initialization Vector (SIV) Authenticated Encryption Using the Advanced Encryption Standard (AES)*. Internet Engineering Task Force, RFC 5297, 2008. https://tools.ietf.org/html/rfc5297. 37

[RFC5487] M. Badra. *Pre-Shared Key Cipher Suites for TLS with SHA-256/384 and AES Galois Counter Mode*. Internet Engineering Task Force, RFC 5487, 2009. https://tools.ietf.org/html/rfc5487. 70

[RFC5489] I. Hajjeh and M. Badra. *ECDHE_PSK Cipher Suites for Transport Layer Security (TLS)*. Internet Engineering Task Force, RFC 5489, 2009. https://tools.ietf.org/html/rfc5489. 70

[RFC5639] M. Locheter and J. Merkle. *Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve generation*. Internet Engineering Task Force, RFC 5639, 2010. https://tools.ietf.org/html/rfc5639. 47

[RFC5647] K. Igoe and J. Solinas. *AES Galois Counter Mode for the Secure Shell Transport Layer Protocol*. Internet Engineering Task Force, RFC 5647, 2009. https://tools.ietf.org/html/rfc5647. 74

[RFC5656] D. Stebila and J. Green. *Elliptic Curve Algorithm Integration in the Secure Shell Transport Layer*. Internet Engineering Task

Force, RFC 5656, 2009. https://tools.ietf.org/html/rfc5656. 72, 75

[RFC5869] H. Krawczyk and P. Eronen. *HMAC-based Extract-and-Expand Key Derivation Function (HKDF)*. Internet Engineering Task Force, RFC 5689, 2010. https://www.rfc-editor.org/rfc/pdfrfc/rfc5869.txt.pdf. 38

[RFC5903] D. Fu and J. Solinas. *Elliptic Curve Groups modulo a Prime (ECP Groups) for IKE and IKEv2*. Internet Engineering Task Force, RFC 5903, 2010. https://tools.ietf.org/html/rfc5903. 77

[RFC6176] S. Turner and T. Polk. *Prohibiting Secure Sockets Layer (SSL) Version 2.0*. Internet Engineering Task Force, RFC 6176, 2011. https://tools.ietf.org/html/rfc6176. 61

[RFC6187] K. Igoe and D. Stebila. *X.509v3 Certificates for Secure Shell Authentication*. Internet Engineering Task Force, RFC 6187, 2011. https://tools.ietf.org/html/rfc6187. 75

[RFC6655] D. McGrew and D. Bailey. *AES-CCM Cipher Suites for Transport Layer Security TLS*. Internet Engineering Task Force, RFC 6655, 2008. https://tools.ietf.org/html/rfc6655. 69, 70

[RFC6668] D. Bider and M. Baushke. *SHA-2 Data Integrity Verification for the Secure Shell (SSH) Transport Layer Protocol*. Internet Engineering Task Force, RFC 4344, 2012. https://tools.ietf.org/html/rfc4344. 74

[RFC6954] J. Merkle and M. Lochter. *Using the Elliptic Curve Cryptography (ECC) Brainpool Curves for the Internet Key Exchange Protocol Version 2 (IKEv2)*. Internet Engineering Task Force, RFC 6954, 2013. https://tools.ietf.org/html/rfc6954. 77

[RFC7027] J. Merkle and M. Lochter. *Elliptic Curve Cryptography (ECC) Brainpool Curves for Transport Layer Security (TLS)*. Internet Engineering Task Force, RFC 7027, 2013. https://tools.ietf.org/html/rfc7027. 71

[RFC7250] P. Wouters, H. Tschofenig, J. Gilmore, and T. Kivinen. *Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)*. Internet Engineering Task Force, RFC 7250, 2014. https://tools.ietf.org/html/rfc7250. 63

[RFC7251] D. McGrew, D. Bailey, M. Campagna, and R. Dugal. *AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS*. Internet Engineering Task Force, RFC 7251, 2014. https://tools.ietf.org/html/rfc7251. 68

[RFC7296] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, and T. Kivinen. *Internet Key Exchange Protocol Version 2 (IKEv2)*. Internet Engineering Task Force, RFC 7296, 2014. https://tools.ietf.org/html/rfc7296. 75, 76

[RFC7427] T. Kivinen and J. Snyder. *Signature Authentication in the Internet Key Exchange Version 2 (IKEv2)*. Internet Engineering Task Force, RFC 7427, 2015. https://tools.ietf.org/html/rfc7427. 79

[RFC7465] A. Popov. *Prohibiting RC4 Cipher Suites*. Internet Engineering Task Force, RFC 7465, 2015. https://tools.ietf.org/html/rfc7465. 64

[RFC756] R. Barnes, M. Thomson, A. Pironti, and A. Langley. *Deprecating Secure Sockets Layer Version 3.0*. Internet Engineering Task Force, RFC 7568, 2015. https://tools.ietf.org/html/rfc7568. 60, 61

[RFC7748] A. Langley, M. Hamburg, and S. Turner. *Elliptic curves for security*. Internet Engineering Task Force, RFC 7748, 2016. https://tools.ietf.org/html/rfc7748. 48

[RFC7905] A. Langley, W. Chang, N. Mavrogiannopoulos, J. Strombergson, and S. Josefsson. *ChaCha20-Poly1305 Cipher Suites for Transport Layer Security (TLS)*. Internet Engineering Task Force, RFC 7905, 2016. https://tools.ietf.org/html/rfc7905. 64, 65, 68, 69, 70

[RFC7914] C. Percival and S. Josefsson. *The SCRYPT Password-Based Key Derivation Function*. Internet Engineering Task Force, RFC 7914, 2016. https://tools.ietf.org/html/rfc7914. 39

[RFC7919] D. Gillmor. *Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS)*. Internet Engineering Task Force, RFC 7919, 2016. https://tools.ietf.org/html/rfc7919. 44, 63, 66, 71

[RFC8017] K. Moriarty, B. Kaliski, J. Jonsson, and A. Rusch. *PKCS #1: RSA Cryptography Specifications Version 2.2*. Internet Engineering Task Force, RFC 8017, 2016. https://datatracker.ietf.org/doc/html/rfc8017. 50, 52, 79

[RFC8018] K. Moriarty, B. Kaliski, and A. Rusch. *PKCS #5: Password-Based Cryptography Specification. Version 2.1*. Internet Engineering Task Force, RFC 8018, 2017. https://tools.ietf.org/html/rfc8018. 38

[RFC8031] Y. Nir and S. Josefsson. *Curve25519 and Curve448 for the Internet Key Exchange Protocol Version 2 (IKEv2) Key Agreement*. Internet Engineering Task Force, RFC 8031, 2016. https://tools.ietf.org/html/rfc8031. 77

[RFC8032] S. Josefsson and I. Liusvaara. *Edwards-Curve Digital Signature Algorithm (EdDSA)*. Internet Engineering Task Force, RFC 8032, 2017. https://www.rfc-editor.org/rfc/pdfrfc/rfc8032.txt.pdf. 169, 170, 172, 173

[RFC8221] P. Wouters, D. Migault, J. Marrsson, Y. Nir, and T. Kivien. *Cryptographic Algorithm Implementation Requirements and Usage Guidance for Encapsulating Security Payload (ESP) and Authentication Header (AH)*. Internet Engineering Task Force, RFC 8221, 2017. https://tools.ietf.org/html/rfc8221. 75, 76

[RFC8247] Y. Nir, T. Kivien, P. Wouters, and D. Migault. *Algorithm Implementation Requirements and Usage Guidance for the Internet Key Exchange Protocol Version 2 (IKEv2)*. Internet Engineering Task Force, RFC 8247, 2017. https://tools.ietf.org/html/rfc8247. 39, 76

[RFC8391] A. Hülsing, D. Butin, S. Gazdag, J. Rijneveld, and A. Mohaisen. *XMSS: extended Merkle signature scheme*. Internet Engineering Task Force, RFC 8391, 2018. https://tools.ietf.org/html/rfc8391. 52, 54, 141, 154, 158

[RFC8422] Y. Nir, S. Josefsson, and M. Pégourié-Gonnard. *Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier*. Internet Engineering Task Force, RFC 8422, 2013. https://tools.ietf.org/html/rfc8422. 66, 71

[RFC8439] Y. Nir and A. Langley. *ChaCha20 and Poly1305 for IETF Protocols*. Internet Engineering Task Force, RFC 8439, 2018. https://tools.ietf.org/html/rfc8439. 35, 64

[RFC8442] J. Mattsson and D. Migault. *ECDHE_PSK with AES-GCM and AES-CCM Cipher Suites for TLS 1.2 and DTLS 1.2*. Internet Engineering Task Force, RFC 8442, 2018. https://tools.ietf.org/html/rfc8442. 70

[RFC8446] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. Internet Engineering Task Force, RFC 8446, 2018. https://tools.ietf.org/html/rfc8446. 61, 62, 65, 67

[RFC8734] L. Bruckert, J. Merkle, and M. Lochter. *Elliptic Curve Cryptography (ECC) Brainpool Curves for Transport Layer Security (TLS) Version 1.3*. Internet Engineering Task Force, RFC 8734, 2020. https://tools.ietf.org/html/rfc8734. 66, 67

[RFC9106] A. Biryukov, D. Dinu, D. Khovratovich, and S. Josefsson. *Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications*. Internet Engineering Task Force, RFC 9106, 2021. https://www.rfc-editor.org/info/rfc9106. 39

[RSA78] R. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978. https://doi.org/10.1145/359340.359342. 17, 43

[RSA12] RSA Labs. *PKCS #1 v2.2: RSA Cryptography Standard*. RSA Laboratories, 2012. https://mpqs.free.fr/h11300-pkcs-1v2-2-rsa-cryptography-standard-wp_EMC_Corporation_Public-Key_Cryptography_Standards_(PKCS).pdf. 50, 52

[SAB+20] P. Schwabe, R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, G. Seiler, and D. Stehle. CRYSTALS-KYBER. Online publication, 2020. https://pq-crystals.org/kyber/index.shtml. 58

[Sch91] C. P. Schnorr. Efficient signature generation by smartcards. *Journal of Cryptology*, (4):161–174, 1991. https://doi.org/10.1007/BF00196725. 166

[Sch21] B. Schoenmakers. *Lecture Notes, Cryptographic Protocols. Version 1.6*. Department of Mathematics and Computer Science, Technical University of Eindhoven, 2021. https://www.win.tue.nl/~berry/CryptographicProtocols/LectureNotes.pdf. 13

[SEC10] SEC. *Recommended Elliptic Curve Domain Parameters*. Standards for Efficient Cryptography Group, Version 2.0, 2010. 73, 75

[Sha79] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979. https://doi.org/10.1145/359168.359176. 27

[Sho97] P. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. https://doi.org/10.1137/S0097539795293172. 12, 48, 106, 157

[Sim97] D. Simon. On the power of quantum computation. *SIAM Journal on Computing*, 26(5):1474–1483, 1997. https://doi.org/10.1137/S0097539796298637. 107

[SJP14] G. Sarath, D. Jinwala, and S. Patel. A survey on elliptic curve digital signature algorithm and its variants. In *Proc. Second International Conference on Computational Science and Engineering, Computer Science & Information Technology*, volume 4, pages 121–136, 2014. https://doi.org/10.5121/csit.2014.4411. 177

[SP800-108] NIST. *Recommendation for Key Derivation Using Pseudorandom Functions*. National Institute of Standard and Technology, Special Publication SP800-108, Revision 1, 2024. https://doi.org/10.6028/NIST.SP.800-108r1-upd1. 38

[SP800-132] NIST. *Recommendation for Password-Based Key Derivation. Part 1: Storage Applications*. National Institute of Standard and Technology, Special Publication SP800-132, Addendum to SP800-132, 2010. https://doi.org/10.6028/NIST.SP.800-132. 38

[SP800-185] NIST. *SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash, and ParallelHash*. National Institute of Standard and Technology, Special Publication, SP800-185, Rev 1 (November 2014 draft), 2016. https://doi.org/10.6028/NIST.SP.800-185. 26, 31, 129

[SP800-186] NIST. *Recommendations for Discrete Logarithm-based Cryptography: Elliptic Curve Domain Parameters*. National Institute of Standard and Technology, Special Publication, SP800-186 (February, 2022), 2023. https://doi.org/10.6028/NIST.SP.800-186. 47, 73, 75, 169, 170, 173

[SP800-208] NIST. *Recommendation for Stateful Hash-Based Signature Schemes*. National Institute of Standard and Technology, Special Publication, SP800-208, 2020. https://doi.org/10.6028/NIST.SP.800-208. 52, 54, 141, 154

[SP800-38A-Add] NIST. *Recommendation for Block Cipher Modes of Operation: Three Variants of Ciphertext Stealing for CBC Mode*. National Institute of Standard and Technology, Special Publication,

Addendum to SP800-38A, 2012. https://doi.org/10.6028/NIST.SP.800-38A-Add. 28

[SP800-38A] NIST. *Recommendation for Block Cipher Modes of Operation. Methods and Techniques*. National Institute of Standards and Technology, Special Publication SP800-38A, March 2001. https://doi.org/10.6028/NIST.SP.800-38A. 28

[SP800-38B] NIST. *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*. National Institute of Standards and Technology, Special Publication SP800-38B, May 2005. https://doi.org/10.6028/NIST.SP.800-38B. 31

[SP800-38C] NIST. *Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality*. National Institute of Standards and Technology, Special Publication SP800-38C, May 2007. https://doi.org/10.6028/NIST.SP.800-38C. 35

[SP800-38D] NIST. *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. National Institute of Standards and Technology, Special Publication SP800-38D, November 2007. https://doi.org/10.6028/NIST.SP.800-38D. 31, 32, 35

[SP800-38E] NIST. *Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices*. National Institute of Standards and Technology, Special Publication SP800-38E, November 2010. https://doi.org/10.6028/NIST.SP.800-38E. 30

[SP800-38F] NIST. *Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping*. National Institute of Standards and Technology, Special Publication SP800-38F, December 2012. https://doi.org/10.6028/NIST.SP.800-38F. 37

[SP800-56A] NIST. *Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography*. National Institute of Standard and Technology, Special Publication, SP800-56A, Rev. 3, 2018. https://doi.org/10.6028/NIST.SP.800-56Ar3. 38, 57, 107

[SP800-56B] NIST. *Recommendation for Pair-Wise Key-Establishment Using Integer Factorization Cryptography*. National Institute of Standard and Technology, Special Publication, SP800-56B, Rev. 2, 2019. https://doi.org/10.6028/NIST.SP.800-56Br2. 38, 107

[SP800-56C] NIST. *Recommendation for Key-Derivation Methods in Key-Establishment Schemes*. National Institute of Standard and Technology, Special Publication, SP800-56C, Rev. 2, 2020. https://doi.org/10.6028/NIST.SP.800-56Cr2. 38

[SP800-90A] NIST. *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*. National Institute of Standard and Technology, Special Publication, SP800-90A, Rev 1, 2015. https://doi.org/10.6028/NIST.SP.800-90Ar1. 84

[Ste21] M. Stern. Re: Diversity of signature schemes. Online publication, 2021. https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/2LEoSpskELs/m/LkUdQ5mKAwAJ. 142

[TR-02102-] BSI. *Cryptographic Mechanisms: Recommendations and Key Lengths, Part 4 - Use of Secure Shell (SSH), version 2022-01*. Bundesamt für Sicherheit in der Informationstechnik, BSI TR-02102-4, 2022/01/24, 2022. https://www.bsi.bund.de/EN/Service-Navi/Publications/TechnicalGuidelines/tr02102/tr02102_node.html. 72

[TR-02102-2] BSI. *Cryptographic Mechanisms: Recommendations and Key Lengths, Part 2 - Use of Transport Layer Security (TLS), version 2022-01*. Bundesamt für Sicherheit in der Informationstechnik, BSI TR-02102-2, 2022/01/24, 2022. https://www.bsi.bund.de/EN/Service-Navi/Publications/TechnicalGuidelines/tr02102/tr02102_node.html. 60

[TR-03111v2.1] BSI. *Elliptic curve cryptography*. Bundesamt für Sicherheit in der Informationstechnik, BSI TR-03111 version 2.1, 2018. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TR03111/BSI-TR-03111_V-2-1_pdf. 52, 167, 180

[vNS+16a] P. Švenda, M. Nemec, P. Sekan, R. Kvašňovský, D. Formánek, D. Komárek, and V. Matyáš. The million-key question – Investigating the origins of RSA public keys. Technical report, Faculty odf Informatics, Masaryk University, Brno, 2016. http://www.fi.muni.cz/reports. 104

[vNS+16b] P. Švenda, M. Nemec, P. Sekan, R. Kvašňovský, D. Formánek, D. Komárek, and V. Matyáš. The million-key question—Investigating the origins of RSA public keys. In *Proc. 25th USENIX Security Symposium (USENIX 16)*, pages

893–910, Austin, TX, 2016. USENIX Association. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/svenda. 104

[VP15] M. Vanhoef and F. Piessens. All your biases belong to us: Breaking RC4 in WPA-TKIP and TLS. In *Proc. 24nd USENIX Security Symposium*, pages 97–112, 2015. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/vanhoef. 64

[Ylö96] T. Ylönen. SSH - secure login connections over the internet. In *Proc. 6th USENIX Security Symposium (USENIX 96)*, pages 37–42, 1996. https://www.usenix.org/legacy/publications/library/proceedings/sec96/full_papers/ylonen/index.html. 72

PUBLICACIONES DE DEFENSA

GOBIERNO DE ESPAÑA MINISTERIO DE DEFENSA SUBSECRETARÍA DE DEFENSA SECRETARÍA GENERAL TÉCNICA SUBDIRECCIÓN GENERAL DE PUBLICACIONES Y PATRIMONIO CULTURAL