Sección de Matemáticas
Universidad de La Laguna

Édgar Pérez Ramos

Estudio e Implementación del Esquema de Firma CRYSTALS-Dilithium

Study and Implementation of the CRYSTALS-Dilithium Signature Scheme

Trabajo Fin de Grado
Grado en Matemáticas
La Laguna, Mayo de 2023

Dirigido por
Pino Caballero Gil

Pino Caballero Gil
Departamento de Ingeniería
Informática y Sistemas
Universidad de La Laguna
38200 La Laguna, Tenerife

Agradecimientos

A mi familia, mis padres, mi hermana y mi tía, por apoyarme siempre y estar en todo y para todo. Gracias por inculcarme el valor del esfuerzo y que no hay que tener miedo a los grandes cambios.

A Amanda, hemos recorrido juntos este camino. No he podido tener mejor compañía. Gracias por todo.

A Gregorio, por hacerme disfrutar las clases de matemáticas, transmitirme su pasión y ser mi amigo.

A Pino, por brindarme la oportunidad de realizar este maravilloso trabajo, por su cercanía y por ser una referente.

A Dani, por ayudarme con parte de este trabajo y poder terminar la implementación.

Por último, me gustaría agradecer a todas aquellas personas que me han ayudado a crecer, a todas esas películas y libros que me han dado refugio en tiempos de tormenta y sobre todo a los tiempos amargos que me han hecho valorar los momentos de felicidad.

Édgar Pérez Ramos
La Laguna, 26 de mayo de 2023

Resumen · Abstract

Resumen

CRYSTALS-Dilithium es uno de los tres esquemas de firma digital incluidos en la tercera ronda de elección de estándar post-cuántico del National Institute of Standards and Technology. Se trata de un algoritmo basado en retículos, que sigue las pautas del conocido esquema de Fiat-Shamir. Concretamente, su seguridad se basa en la dificultad del problema de encontrar vectores más cortos en retículos. En este trabajo se presenta una introducción a los conceptos que sustentan el protocolo, y una implementación del mismo con un objetivo claramente didáctico.

Palabras clave: *Criptografía post-cuántica – Retículos – LWE – CRYSTALS-Dilithium*

Abstract

CRYSTALS-Dilithium is one of the three digital signature schemes included in the third round of post-quantum standard selection by the National Institute of Standards and Technology. It is a lattice-based algorithm that adheres to the well-known Fiat-Shamir scheme guidelines. Specifically, its security is based on the difficulty of the problem of finding shorter vectors in lattices. This paper provides an introduction to the underlying concepts of the protocol and presents its implementation with a clear educational objective.

Keywords: *Post-Quantum Cryptography – Lattices – LWE – CRYSTALS-Dilithium*

Contenido

Introducción

La criptografía ha existido desde tiempos antiguos. Se han encontrado mensajes cifrados en inscripciones egipcias de hace más de 4.000 años, y los antiguos griegos utilizaron la transposición y la sustitución de letras para proteger sus comunicaciones. Durante la Edad Media, los monjes copistas desarrollaron técnicas para ocultar información en manuscritos, y los soldados utilizaban claves secretas para enviar mensajes en el campo de batalla.

En tiempos modernos, la criptografía se ha vuelto aún más importante debido a la creciente cantidad de información digital que se transmite y almacena. Los algoritmos criptográficos modernos se basan en complejas operaciones y en la teoría de la información para proteger la confidencialidad, integridad y autenticidad de la información.

Hoy en día, la criptografía se aplica en muchos ámbitos, como la seguridad informática, la banca, la medicina, las comunicaciones militares, entre otros. La criptografía también se utiliza para proteger la privacidad de las comunicaciones en línea y para garantizar la seguridad de las transacciones financieras en internet.

En la actualidad, uno de los mayores desafíos para la criptografía es la llegada de los ordenadores cuánticos, que podrían ser capaces de romper muchos de los algoritmos criptográficos convencionales que se utilizan hoy en día. Por esta razón, en el año 2016 arrancó la carrera de la criptografía post-cuántica. Ese año el National Institute of Standards and Technology (NIST) inició un proceso de selección de esquemas criptográficos que pudieran ser resistentes a los ordenadores cuánticos. A finales de 2017 se publicaron los algoritmos que habían pasado la primera fase de la convocatoria, un total de 69. Más tarde, en 2019 tras aplicar un cribado exhaustivo de los 69 iniciales quedaron 26. Durante esa ronda se llevaron a cabo pruebas rigurosas para evaluar la seguridad y el rendimiento de cada uno de los algoritmos, así como para identificar posibles

vulnerabilidades y mejorar la seguridad de los algoritmos. En la tercera ronda de evaluación en 2018, se seleccionaron 15 candidatos finales para continuar en el proceso de estandarización. Durante esta ronda, se llevaron a cabo pruebas adicionales y se trabajó con la comunidad criptográfica para identificar posibles vulnerabilidades y mejorar la seguridad de los algoritmos preseleccionados.
Finalmente, según [1] en julio de 2022, el NIST anunció los algoritmos seleccionados como estándares finales, diferenciando entre:

- Cifrado: CRYSTALS-Kyber.
- Firma digital: CRYSTALS-Dilithium, FALCON y SPHINCS+.

En este trabajo se presenta un estudio del esquema de firma digital CRYSTALS-Dilithium [2] [3], basado en la estructura matemática de los retículos [4].

Se espera que Dilithium se utilice ampliamente en la industria y el gobierno para proteger la autenticidad de los documentos digitales y los mensajes en sistemas informáticos y de comunicaciones durante las próximas décadas.

Este trabajo se estructura de la siguiente forma. En el capítulo 1 se analiza en profundidad el concepto de retículo, trabajando algunas propiedades algebraicas y geométricas. En el capítulo 2 se estudian los problemas computacionales asociados a los mismos, como el problema del vector más corto y el problema de aprendizaje con errores. En el capítulo 3 se analiza el esquema de firma de CRYSTALS-Dilithium, incluyendo algunos algoritmos complementarios y sus bases matemáticas. En el capítulo 4 se desarrolla una implementación del esquema de firma y se cierra el trabajo con algunas conclusiones y trabajos futuros. Finalmente, en el apéndice se incluye la referencia a un artículo producto de este trabajo, que ha sido aceptado a un congreso nacional y que será presentado en las próximas semanas.

1

Retículos

Antes de comenzar a estudiar en profundidad los retículos se deben controlar algunos conceptos algebraicos previos que van a ayudar a conocer y entender mejor los cimientos de este objeto matemático y sus propiedades.

1.1. Preliminares

1.1.1. Grupos

Definición 1.1.1.1 *Sea G un conjunto no vacío y $\#$ una operación binaria definida en G. Se dice que el par $(G, \#)$ es un grupo si se cumplen las siguientes propiedades:*

1. *Propiedad asociativa.* $\forall a, b, c \in G \ : \quad (a \# b) \# c = a \# (b \# c)$.
2. *Existencia del elemento neutro.* $\exists\, e \in G \ : \ e \# u = u \# e, \qquad \forall u \in G$.
3. *Existencia del elemento opuesto.* $\forall\, a \in G, \exists\, b \in G \ : \ a \# b = e = b \# a$.

Si se da la propiedad conmutativa respecto a la operación $\#$ se habla de un *grupo abeliano*. Además, en general se suele denotar las operaciones de la siguiente manera: $(G, +)$ en notación aditiva y $(G, \cdot)$ en notación multiplicativa.

Definición 1.1.1.2 *Sea $(G, \cdot)$ es un grupo y $H \subset G$, se dice que $(H, +)$ es un subgrupo de $(G, +)$ si también tiene estructura de grupo con la misma operación. En caso de que ocurra se denota:*

$$(H, +) \leq (G, +) \qquad o \qquad H \leq G$$

1.1.2. Anillos

En esta sección se recordará la definición de anillo y de cuerpo. El objetivo que se persigue es adquirir las herramientas necesarias para abordar los espacios vectoriales y, más adelante, los retículos.

Definición 1.1.2.1 *Sea A un conjunto no vacío dotado de dos operaciones binarias. Se dice que la terna $(A, +, \cdot)$ es un anillo si se cumplen las siguientes condiciones:*

- *Respecto a la operación $+$:*
  1. *Propiedad asociativa.* $\forall a, b, c \in A \ : \ (a + b) + c = a + (b + c)$.
  2. *Propiedad conmutativa.* $\forall a, b \in A \ : \ a + b = b + a$.
  3. *Existencia del elemento neutro.* $\exists\, e \in A \ : \ e + u = u = u + e, \qquad \forall u \in A$. *Se denota al elemento neutro por* 0_A.
  4. *Existencia del simétrico.* $\forall\, a \in A, \exists\, b \in A \ : \ a + b = e = b + a$. *Al elemento simétrico de $a \in A$ se le denota como $-a$ y se le conoce por el opuesto de a.*
- *Respecto a la operación $\cdot$:*
  1. *Propiedad asociativa.* $\forall a, b, c \in A \quad : \quad (a \cdot b) \cdot c = a \cdot (b \cdot c)$.
  2. *Propiedad distributiva.* $\forall a, b, c \in A \quad : \quad (a + b) \cdot c = a \cdot c + b \cdot c$.
  3. *En caso de existir,* $\exists\, e \in A : e \cdot a = a \cdot e = a$. *El elemento neutro del producto se denota por* 1_A.

En el caso de que se cumpla la propiedad conmutativa y exista el elemento neutro de la operación $\cdot$ se habla de un anillo conmutativo y unitario.

Para explicar qué es un cuerpo, se requiere tener en cuenta la noción de unidad. Es importante recordar cómo se define:

Definición 1.1.2.2 *Sea $(A, +, \cdot)$ un anillo unitario. Se dice que $a \in A \setminus 0_A$ es una unidad de A si $\exists\, b \in A$:*

$$a \cdot b = 1_A = b \cdot a$$

Al elemento $b \in A$ se le llama inverso de $a \in A$ y se escribe: $b = a^{-1}$. Además, el conjunto de las unidades de A se denota de la siguiente manera:

$$A^* = \{a \in A \setminus \{0_A\} \ : \ a \cdot b = 1_A = b \cdot a, \ \exists\, b \in A\}$$

Definición 1.1.2.3 *Sea $(A, +, \cdot)$ un anillo conmutativo y unitario. Se dice que A es un cuerpo si $A^* = A \setminus \{0_A\}$.*

Para abordar los conjuntos cocientes y los problemas relacionados con los retículos, resulta imprescindible contar con la definición de los ideales.

Definición 1.1.2.4 *Sea A un anillo conmutativo e I un subconjunto de A no vacío. Se dice que I es un ideal si:*

1. $0_A \in I$.
2. $\forall a \in A, \forall b \in I, \ \ a \cdot b \in I$.
3. $\forall a, b \in I, \ \ a - b \in I$.

Definición 1.1.2.5 *Sean A un anillo conmutativo y unitario e I un ideal de A. En A se define la siguiente relación binaria:*

$$\forall a, b \in A \,:\, a \sim_I b \iff a - b \in I$$

La relación $\sim_I$ es una relación de equivalencia. Se denota por A/I al conjunto cociente asociado a $\sim_I$, es decir:

$$A/I = \{a + I : a \in A\}$$

Además A/I tiene estructura de anillo conmutativo y unitario con las siguientes operaciones:

- *Adición.*

$$\begin{aligned} +\colon A/I \times A/I &\longrightarrow A/I \\ (a+I, b+I) &\longmapsto (a+b)+I, \end{aligned}$$

- *Producto.*

$$\begin{aligned} \cdot\colon A/I \times A/I &\longrightarrow A/I \\ (a+I, b+I) &\longmapsto (a\cdot b)+I, \end{aligned}$$

Ejemplo 1 *$A = \mathbb{Z}$ y $I = (3)$, es decir, I es el conjunto de los múltiplos de 3. Por tanto: $A/I = \{0 + I, 1 + I, 2 + I\}$.*
Este concepto adquiere importancia cuando se trabaja con ideales y anillos de polinomios del estilo $\mathbb{Z}_p$, con p un número primo.

1.1.3. Espacios vectoriales.

Definición 1.1.3.1 *Sea V un espacio vectorial sobre un cuerpo K, con V distinto de vacío y dotado con dos operaciones:*

- *Adición.*

$$\begin{aligned} +\colon V \times V &\longrightarrow V \\ (u, v) &\longmapsto u + v, \end{aligned}$$

es una operación interna que cumple las siguientes condiciones:
1. *Conmutativa. $\forall u, v \in V,\ u + v = v + u$.*
2. *Asociativa. $\forall u, v, w \in V,\ (u + v) + w = u + (v + w)$.*
3. *Existencia del elemento neutro. $\exists\, e \in V \ :\ e + u = u + e = u,\ \forall u \in V$.*
4. *Existencia del opuesto. $\forall u \in V, \exists\, v \in V \ :\ u + v = e = v + u$.*

- *Producto.*

$$\begin{aligned} \cdot : K \times V &\longrightarrow V \\ (a, v) &\longmapsto a \cdot v, \end{aligned}$$

es una operación externa tal que:
1. *Asociativa.* $\forall a, b \in K, \forall u \in V \ : \ a \cdot (b \cdot u) = (a \cdot b) \cdot u.$
2. *Existencia del elemento neutro.* $\exists\, e \in K, \forall u \in V \ : \ e \cdot u = u.$
3. *Distributiva.* $\forall a \in K, \forall u, v \in V \ : \ a \cdot (v + u) = a \cdot v + a \cdot u.$

A la terna $(V, +, \cdot)$ se le conoce como espacio vectorial sobre el cuerpo K.

Definición 1.1.3.2 *Sea V un espacio vectorial sobre un cuerpo K y un subconjunto $B \subset V$ es una base si:*

1. *B es linealmente independiente.*
2. *B es un sistema de generadores de V.*

Las bases revelan la estructura de los espacios vectoriales de una manera concisa. Una base es el menor conjunto (finito o infinito) que genera V, con $B = \{v_i\}_{i \in I}$ y $B \subseteq V$. Esto significa que cualquier vector v puede ser expresado como una combinación lineal de elementos de la base.

$$\forall u \in V, \quad u = \sum_{i=1}^{n} a_i \cdot v_i, \quad a_i \in K$$

Es posible abordar la dimensión del espacio vectorial utilizando la referencia bibliográfica proporcionada por [5].

- Si B es una base finita, es decir tiene n vectores, entonces $dim(V) = n$.
- Si B es una base infinita, entonces $dim(V) = \infty$.

Se ha realizado un breve repaso de las herramientas necesarias para estudiar los retículos. Ahora, los retículos son objetos geométricos que pueden describirse intuitivamente como los puntos de intersección de una malla, similar a la red infinita de una portería de fútbol. Esta malla no necesariamente es ortogonal y puede tener un número arbitrario de dimensiones, (ver Fig. 1.1).

1.2. Retículos

1.2.1. Primeras definiciones

Definición 1.2.1.1 *Sea V un espacio vectorial sobre K, con K cuerpo y $B = \{v_1, v_2, \ldots, v_n\}$ una base de un subespacio vectorial de V, y A un anillo contenido en K. Entonces el retículo $\mathcal{L} \subset V$ generado por $\{v_1, v_2, \cdots, v_n\}$ es el conjunto:*

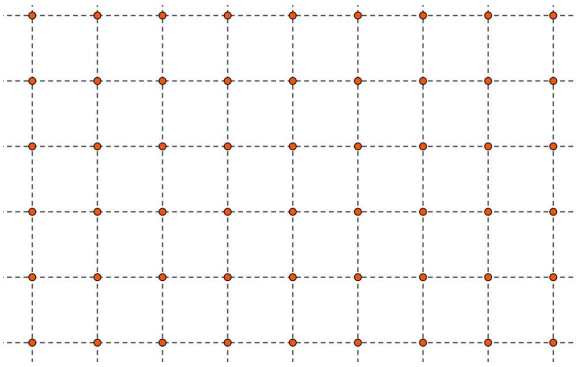


Figura 1.1. Retículo ortonormal sobre el plano euclídeo

$$\mathcal{L}(v_1, v_2, \ldots, v_n) = \{ \sum_{i=1}^{n} a_i \cdot v_i : a_i \in A \}$$

En el resto del trabajo se considera $V = \mathbb{R}^m$ y $\mathbb{Z}$ el anillo A, es decir:

$$\mathcal{L}(v_1, v_2, \ldots, v_n) = \{ \sum_{i=1}^{n} a_i \cdot v_i : a_i \in \mathbb{Z} \}$$

Por tanto, un retículo siempre se puede generar a partir de una base del espacio vectorial en el que se defina, mediante todas las combinaciones lineales de elementos de esa base. De hecho, diferentes conjuntos de vectores pueden generar el mismo retículo o, en otras palabras, un mismo retículo puede ser definido a partir de varias bases diferentes.
Nótese que en el rango del retículo es n, pues hay n vectores linealmente independientes y su dimensión es m, ya que $V = \mathbb{R}^m$.

Se denota $B = \{v_1, v_2, \ldots, v_n\}$ a la base del retículo que puede ser representado de la forma siguiente: $B = [v_1, v_2, \ldots, v_n] \in \mathbb{R}^{m \times n}$, que nos da lugar a una representación equivalente:

$$\mathcal{L}(B) = \{ Bx : x \in \mathbb{Z}^n \}$$

Se puede enunciar el siguiente el resultado.

Proposición 1.2.1.1 *Dado $B \in \mathbb{R}^{m \times n}$, con columnas linealmente independientes. $\mathcal{L} \subset \mathbb{R}^n$ es de rango máximo si y solo si $< B > = \{ Bx : x \in \mathbb{R}^n \} = \mathbb{R}^n$. Es decir,*

$$n = m \iff < B > = \{ Bx : x \in \mathbb{R}^n \}$$

Nota 1.2.1 *De esta proposición se puede deducir que el rango del retículo se caracteriza como la dimensión del espacio que genera su base. O de otra manera,*

$$rango(\mathcal{L}(B)) = dim(< B >)$$

Definición 1.2.1.2 *Sean $\mathcal{L}' = \mathcal{L}'(B')$ y $\mathcal{L} = \mathcal{L}(B)$ retículos generados por las bases B' y B respectivamente. Si $B' \subset B$ entonces se dice que $\mathcal{L}'$ es un subretículo de $\mathcal{L}$.*

Ejemplo 2 *Sea el retículo $\mathcal{L} = \mathcal{L}((1,0,0),(0,1,0),(0,0,1))$. Entonces un subretículo de $\mathcal{L}$ puede ser $\mathcal{L}' = \mathcal{L}((1,0,0),(0,1,0))$, ya que:*

$$B' = \{(1,0,0),(0,1,0)\} \subsetneq B = \{(1,0,0),(0,1,0),(0,0,1)\}$$

Ejemplo de un no subretículo:

$$\mathcal{L}'' = ((\frac{1}{2},0),(0,\frac{1}{2})) \nsubseteq \mathcal{L} = ((1,0),(0,1))$$

Puesto que:

$$\{(\frac{1}{2},0),(0,\frac{1}{2})\} \nsubseteq \{(1,0),(0,1)\},$$

ya que no se puede expresar $(\frac{1}{2},0)$ como combinación lineal de de la base B.

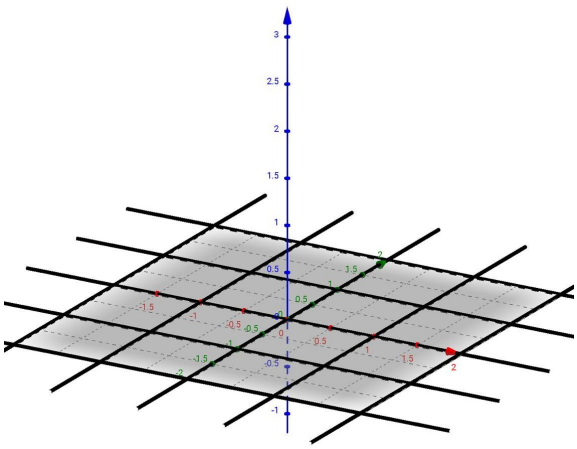


Figura 1.2. Subretículo de dimensión 2 en el espacio

Para construir un subretículo basta con construirlo a partir de una base en la que se hayan suprimido vectores de la base original.

1.2.2. Bases y caracterización algebraica

En general existe más de una base que genera el espacio, es decir, B y B' generan el mismo retículo. Se denota al retículo por $\Lambda = \mathcal{L}(B)$ sin hacer referencia a ninguna base en particular.

Supongamos que $B = \{v_1, v_2, \ldots, v_n\}$ es una base de nuestro retículo $\mathcal{L}(B)$ y sea $B' = \{w_1, w_2, \ldots, w_n\}$ otro conjunto de vectores de V. Entonces se puede escribir como combinación lineal cada $w_j \in B'$ con los elementos de V. Es decir:

$$\begin{cases} w_1 = \alpha_{1,1}v_1 + \alpha_{1,2}v_2 + \ldots + \alpha_{1,n}v_n \\ w_2 = \alpha_{2,1}v_1 + \alpha_{2,2}v_2 + \ldots + \alpha_{2,n}v_n \\ \vdots \\ w_n = \alpha_{n,1}v_1 + \alpha_{n,2}v_2 + \ldots + \alpha_{n,n}v_n \end{cases} \tag{1.1}$$

Sabiendo que $\alpha_{i,j} \in \mathbb{Z}$, $\forall i, j \in \{1, 2, \ldots, n\}$. De esta forma, la matriz que define ese sistema de ecuaciones, llamada matriz de cambio de base, es:

$$A = \begin{pmatrix} \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,n} \\ \alpha_{2,1} & \alpha_{2,2} & \cdots & \alpha_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{n,1} & \alpha_{n,2} & \cdots & \alpha_{n,n} \end{pmatrix} \tag{1.2}$$

Proposición 1.2.2.1 *Dada una matriz A con coeficientes enteros y A^{-1} su matriz inversa. Entonces A^{-1} es de coeficientes enteros si y solo si $det(A) = \pm 1$.*

Demostración.
Se procede de izquierda a derecha:
Hipótesis: $A, A^{-1} \in \mathcal{M}_{n\times n}(\mathbb{Z})$.
Tesis: $det(A) = \pm 1$.

$$1 = det(I) = det(A \cdot A^{-1}) = det(A)det(A^{-1})$$

Pero $det(A), det(A^{-1}) \in \mathbb{Z}$, pues tienen entradas enteras.
Además se sabe que las únicas unidades en $\mathbb{Z}$ son $\pm 1 \Rightarrow det(A) = \pm 1$.

Ahora se avanza en dirección opuesta.
Hipótesis: $det(A) = \pm 1$ y $A \in \mathcal{M}_{n\times n}(\mathbb{Z})$.
Tesis: $A^{-1} \in \mathcal{M}_{n\times n}(\mathbb{Z})$.
En primer lugar, se sabe que:

$$A^{-1} = \frac{1}{det(A)} \cdot (A^*)^t \tag{1.3}$$

Por hipótesis $det(A) = \pm 1 \Rightarrow A^{-1} = \pm(A^*)^t$.
Por definición, si $A \in \mathcal{M}_{n\times n}(\mathbb{Z}) \Rightarrow A^* \in \mathcal{M}_{n\times n}(\mathbb{Z})$. Entonces:

$$A^{-1} = \pm(A^*)^t \in \mathcal{M}_{n\times n}(\mathbb{Z})$$

■

Definición 1.2.2.1 *Se define como matriz unimodular a la matriz* $U \in \mathcal{M}_{n\times n}(\mathbb{Z})$ *si* $det(U) = \pm 1$.

Proposición 1.2.2.2 *Dadas dos bases* B, B' *de un retículo* Λ, *existe* $U \in \mathcal{M}_{n\times n}(\mathbb{Z})$ *unimodular tal que* $B' = BU$.

Demostración. Sean $B' = [w_1, w_2, \ldots, w_n]$ y $B = [v_1, v_2, \ldots, v_n]$.
Entonces es fácil ver de Eq. 1.2 que:

$$B' = B \cdot A^t = [v_1, v_2, \ldots, v_n] \cdot \begin{pmatrix} \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,n} \\ \alpha_{2,1} & \alpha_{2,2} & \cdots & \alpha_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{n,1} & \alpha_{n,2} & \cdots & \alpha_{n,n} \end{pmatrix}^t$$

Como $det(A) = \pm 1 \Rightarrow det(A^t) = \pm 1$, luego A^t es unimodular. Por tanto,

$$B' = BU$$

■

Nota 1.2.2.1 *Así, para descubrir si dos bases generan el mismo retículo solo hace falta descubrir la correspondiente matriz unimodular* U. *De hecho, es suficiente comprobar que* $B^{-1}B'$ *es unimodular.*

Ejemplo 3 *Considérese las bases:* $B' = \{(1,0,0),(0,1,0),(0,0,1)\}$ *y* $B = \{(1,-1,1),(0,1,1),(0,0,-1)\}$. *Entonces la matriz de cambio de base es:*

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 2 & 1 & -1 \end{pmatrix}$$

Dado que esta matriz es triangular inferior, el cálculo de su determinante es instantáneo, $det(A) = -1$. *Por tanto,* A *es unimodular, y según la Prop. 1.2.2.1, se sabe que* A^{-1} *es también una matriz de coeficientes enteros:*

$$A^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 1 & 1 & -1 \end{pmatrix}$$

1.2.3. Definición alternativa

Existe una definición alternativa para los retículos, pero antes es necesario ver dos definiciones previas.

Definición 1.2.3.1 *Sea $S \subset \mathbb{R}^n$, se dice que S es un subconjunto discreto si:*

$$\exists\, \epsilon > 0 \ : \ \forall x \in S,\ S \cap \{\, w \in \mathbb{R}^n :\| \ x - w \ \| < \epsilon \,\} \ = \ \{x\}$$

Para el caso de un retículo:

$$\exists\, \epsilon > 0,\ \ \forall v \in \Lambda \ \ : \ \ S \cap B(v, \epsilon) \ \ = \ \ \{v\}$$

Para el caso de un retículo $\mathcal{L}$, la definición anterior implica que es posible tomar una bola de radio ϵ centrado en un punto del retículo de forma que dentro de esa bola únicamente se encuentre ese punto del retículo (ver Fig. 1.3):

$$\exists\, \epsilon > 0,\ \forall v \in \mathcal{L} \ : \ S \cap B(x, \epsilon) \ = \ \{x\} \tag{1.4}$$

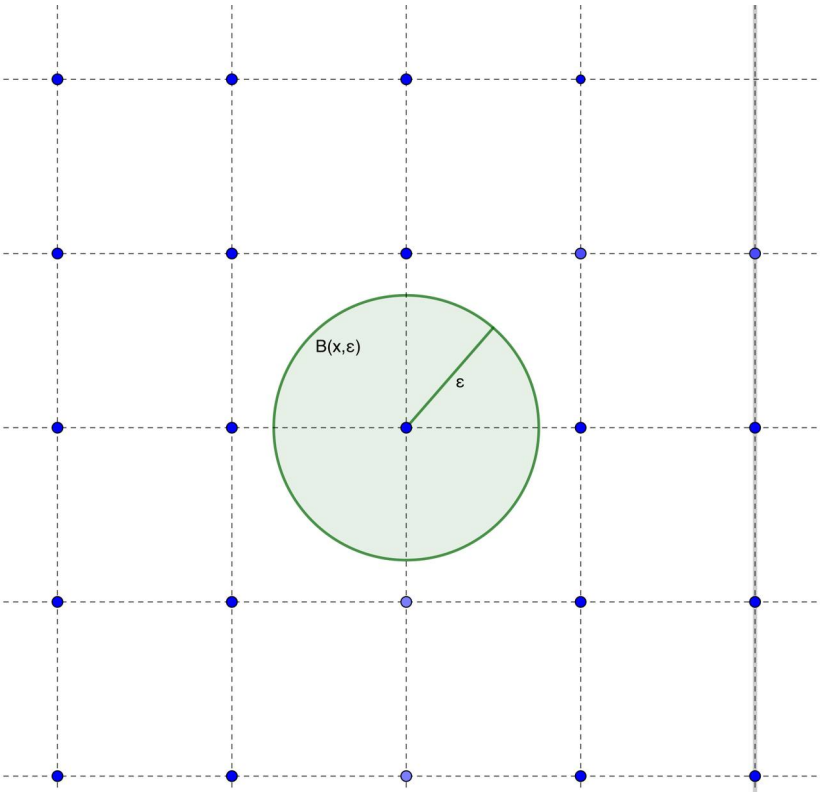


Figura 1.3. Ilustración de un conjunto discreto

Ejemplo 4 *A continuación se dan ejemplos de retículos.*

1. *El conjunto* $\{0\}$.
2. $\mathbb{Z}$ *es un subgrupo aditivo de* $\mathbb{R}$*, por lo que es un retículo. De igual manera, se sabe que el producto cartesiano de un número finito de grupos es también grupo, y pasa lo mismo con los conjuntos discretos. Por tanto,* $\mathbb{Z}^m$ *también es un retículo.*
3. *El conjunto de los números pares,* $2\mathbb{Z}$ *es un grupo aditivo, y además discreto, pues bastaría con tomar una bola de radio* $r < 2$*. Nuevamente esta caracterización se puede generalizar a* $k\mathbb{Z}$*, con* $k \in \mathbb{Z}$*.*

1.2.4. Caracterización geométrica y determinante

Definición 1.2.4.1 *Dados* $v_1, v_2, \ldots, v_n$ *vectores linealmente independientes, se define el paralelepípedo fundamental como:*

$$\mathcal{P}(B) \;=\; \mathcal{P}(v_1, v_2, \ldots, v_n) \;=\; \{ \sum_{i=1}^{n} x_i v_i \;:\; x_i \in [\frac{-1}{2}, \frac{1}{2}) \}$$

En la imagen de la Fig. 1.4 se observa $\mathcal{P}((1,0),(0,1))$.

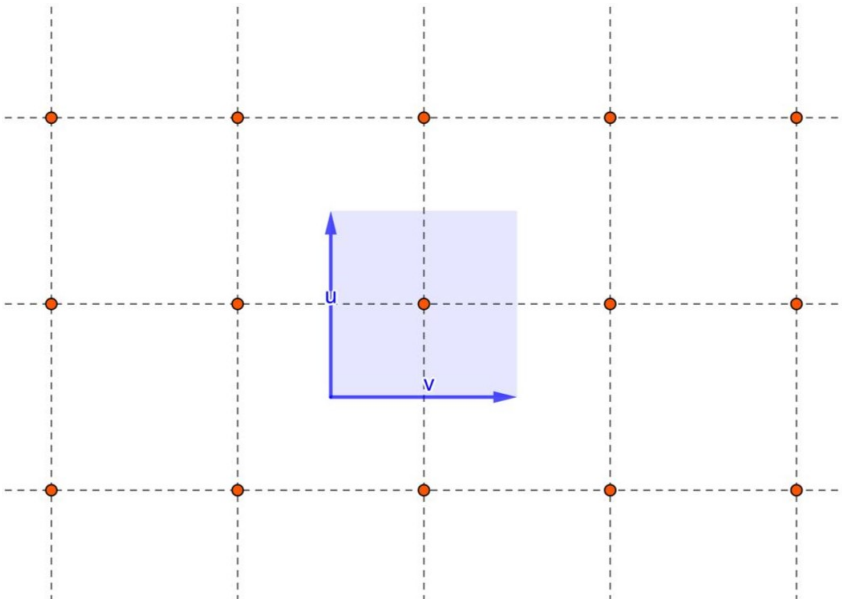


Figura 1.4. $\mathcal{P}((1,0),(0,1))$, con $v = (1,0)$ y $u = (0,1)$

Como se puede apreciar, el paralelepípedo fundamental es la región semiabierta delimitada por los vectores $v_1, v_2, \ldots, v_n$. Se tiene el siguiente resultado con el que se puede teselar con nuestro paralelepípedo todo el espacio.

Proposición 1.2.4.1

$$\bigcup_{v\in\mathcal{L}}(v+\mathcal{P}(B))=\mathbb{R}^n$$

Demostración. Sea $p\in\mathbb{R}^n$. Se puede escribir p de la siguiente manera:

$$p \;=\; \sum_{i=1}^{n}x_iv_i \;=\; \sum_{i=1}^{n}[x_i]v_i+\sum_{i=1}^{n}(x_i-[x_i])v_i,$$

donde $[x_i]$ es x_i redondeado tomando la parte entera Por ejemplo:

$$x_i=1.9\Rightarrow[x_i]=2,\qquad x_i=1.2\Rightarrow[x_i]=1$$

Además,

$$\frac{-1}{2}\leq a-[a]<\frac{1}{2}$$

Por tanto,

1. $\sum_{i=1}^{n}[x_i]v_i\in\mathcal{L}$
2. $\sum_{i=1}^{n}(x_i-[x_i])v_i\in\mathcal{P}(B)$

De lo que se concluye que:

$$p=\underbrace{\sum_{i=1}^{n}[x_i]v_i}_{\in\mathcal{L}}+\underbrace{\sum_{i=1}^{n}(x_i-[x_i])v_i}_{\in\mathcal{P}(B)}\Rightarrow\mathbb{R}^n=\bigcup_{v\in\mathcal{L}}(v+\mathcal{P}(B))$$

Solo queda ver que no se superponen los paralelepípedos fundamentales. Supongamos que $(v+\mathcal{P}(B))\cap(w+\mathcal{P}(B))\neq\varnothing$. Entonces para ciertos $\alpha,\beta\in\mathcal{P}(B)$:

$$v+\alpha=w+\beta\Rightarrow v-w=\beta-\alpha$$

Dado que $v-w$ es una combinación lineal entera de vectores y $\beta-\alpha$ es una combinación lineal que vive en $(-1,1)$, pues $\alpha,\beta\in\mathcal{P}(B)$. Entonces la única opción es que: $v-w=0\Rightarrow v=w$.

■

Definición 1.2.4.2 *Dado un retículo $\mathcal{L}\subset\mathbb{R}^m$ se define su determinante denotado como $det(\mathcal{L})$ como el volumen $n-$dimensional de su paralelepípedo fundamental.*
Esta cantidad es invariante, ya que no depende de la base escogida. En caso de que el retículo sea de rango máximo $(n=m)$ se tiene que:

$$det(\mathcal{L})=|det(B)|$$

Se presenta a continuación un resultado de particular interés:

Proposición 1.2.4.2

$$\mathbb{Z}^n/\mathcal{L} \;=\; vol(\mathcal{P}(B)) \;=\; del(\mathcal{L})$$

Se puede encontrar la demostración en [6].

Una forma alternativa de calcular el determinante de un retículo es la siguiente.

Proposición 1.2.4.3 *Dado un retículo $\mathcal{L} \subset \mathbb{R}^n$ de rango máximo con base B se tiene que:*

$$det(\mathcal{L}) = \prod_{i=1}^{n} \parallel b_i^* \parallel$$

donde $B^* = [b_1^*, b_2^*, \ldots, b_n^*]$ es la base de Gram-Schmidt correspondiente.

Demostración. A continuación se describe el proceso de ortogonalización de Gram-Schmidt de manera matricial:

$$B = [b_1, b_2, \ldots, b_n] = [b_1^*, b_2^*, \ldots, b_n^*] \cdot \begin{pmatrix} 1 & \mu_{21} & \mu_{31} & \cdots & \mu_{n1} \\ 0 & 1 & \mu_{32} & \cdots & \mu_{n2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix} =$$

$$= [\frac{b_1^*}{\parallel b_1^* \parallel}, \ldots, \frac{b_n^*}{\parallel b_n^* \parallel}] \cdot \begin{pmatrix} \parallel b_1^* \parallel & \mu_{21} \cdot \parallel b_1^* \parallel & \mu_{31} \cdot \parallel b_1^* \parallel & \cdots & \mu_{n1} \cdot \parallel b_1^* \parallel \\ 0 & \parallel b_2^* \parallel & \mu_{32} \cdot \parallel b_2^* \parallel & \cdots & \mu_{n2} \cdot \parallel b_2^* \parallel \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \parallel b_n^* \parallel \end{pmatrix}$$

Por lo que:

$$B = B^*T \Rightarrow det(B) = det(B^*)det(T)$$

Como los vectores $\frac{b_i^*}{\|b_i^*\|}$ son ortonormales, el determinante de la matriz de dichos vectores es 1 o -1. Además, al ser la segunda matriz triangular superior se obtiene:

$$det(\mathcal{L}) = |det(B)| = \prod_{i=1}^{n} \parallel b_i^* \parallel$$

■

Proposición 1.2.4.4 *Para cualquier retículo de base $B \in \mathbb{R}^{m \times n}$:*

$$det(\mathcal{L}) = \sqrt{det((B^t)B)}$$

Demostración. Del resultado anterior se deduce que:

$$B = B^*T$$

con T una matriz triangular superior con unos en la diagonal. Entonces:

$$\sqrt{det((B^t)B)} = \sqrt{det(((B^*T)^t)B^*T)} = \sqrt{det(T^tB^*B^*T)} =$$

$$= \sqrt{det(T^*) \cdot det(B^{*t}B^*)det(T)}$$

Por hipótesis $det(T) = 1$, luego:

$$det(B^{*t}B^*) = \prod_{i=1}^{n} < b_i^*, b_i^* > = \prod_{i=1}^{n} \parallel b_i^* \parallel^2 =$$

$$= (\prod_{i=1}^{n} \parallel b_i^* \parallel)^2 = (det(\mathcal{L}))^2$$

Por tanto,

$$det(\mathcal{L}(B) = \sqrt{det(B^tB)}$$

■

Nota 1.2.4.1 *Si B es una matriz cuadrada invertible, entonces $det(B) = det(B^t)$, por lo que $det(\mathcal{L}) = |det(B)|$.*

Ejemplo 5 *Tomando la base $B = \{(1, 0, 0), (1, 2, 3)\}$, si se calcula el producto B^tB, queda una matriz A de rango 2×2, que es:*

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 0 & 2 \\ 0 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 14 \end{pmatrix} \tag{1.5}$$

Por consiguiente, $det(A) = 13$, y $det(\mathcal{L}) = \sqrt{13}$.

2

Problemas asociados a los retículos

En este capítulo se estudian algunos problemas basados en retículos, como son:

1. El problema del vector más corto.
2. El problema del vector más cercano.
3. El problema de aprendizaje con errores (*Learning With Errors, LWE*).

Además se mencionan sus respectivas complejidades computacionales asociadas, todo ello para justificar porqué se escogen estos problemas para CRYSTALS-Dilithium.

2.1. Cotas del vector más corto

Definición 2.1.1 *Dado un retículo $\mathcal{L}$ se define la distancia mínima de $\mathcal{L}$ como:*

$$\lambda_1(\mathcal{L}) \;=\; \min_{v\in\lambda\setminus\{0\}} \| v \| = \min_{x,y\in\mathcal{L}\,:\,x\neq y} \| x-y \| \tag{2.1}$$

A continuación se da la definición de mínimos sucesivos, que ayuda a caracterizar el problema del vector más corto de un retículo.

Definición 2.1.2 *Dado un retículo $\mathcal{L} \subset \mathbb{R}^m$ de rango n, se define para todo $1 \leq i \leq n$,*

$$\lambda_i(\mathcal{L}) \;=\; inf\{r \in \mathbb{R}\},$$

dónde $B(0,r)$ contiene al menos i vectores que son linealmente independientes.

Ejemplo 6 *Supongamos que se tienen dos vectores $u = (1,0)$ y $v = (0,3)$. Si se quisiera calcular los mínimos sucesivos en este caso, se tiene que calcular λ_1 y λ_2, pues solo hay vectores.*

Las bolas $B(0,r)$ que contienen exclusivamente a u, son todas aquellas que tienen radio $r \in [1,3)$, pues u y v son linealmente independientes y solo puede haber un vector dentro. Por tanto, como se quiere calcular el ínfimo:

$$\lambda_1 = 1$$

No obstante, la bola $B(0,r)$*, con* $r = 2$ *contiene también a* u*, sin embargo,* $r \neq \lambda_1$*.*

De manera similar, las bolas $B(0,r)$ *que contienen a los dos vectores son todas aquellas que tienen un radio* $r \geq 3$*, por lo que:*

$$\lambda_2 = 3$$

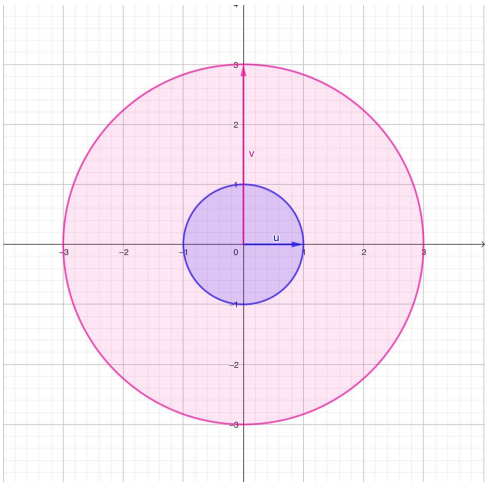


Figura 2.1. El radio de la bola más pequeña que se ha trazado es λ_1

La Fig. 2.1 está inspirada en [7].

Teorema 2.1.1 *Sea* B *la base de un retículo de rango* n *y* B^* *la base de Gram-Schmidt correspondiente. Entonces:*

$$0 \;<\; \min_{i\in\{1,2,\ldots,n\}} \| b_i^* \| \;<\; \lambda_1(\mathcal{L}(B))$$

Se puede consultar la demostración en [6]. A continuación para poder acotar superiormente el vector más corto se enuncia y se demuestra el teorema del Cuerpo Convexo de Minkowski.

Teorema 2.1.2 *Para todo retículo de rango máximo* $\mathcal{L} \subset \mathbb{R}^n$*, dado un conjunto* $S \subset \mathbb{R}^n$ *acotado, convexo y simétrico con* $Vol(S) > 2^n \cdot det(\mathcal{L})$*,* S *contiene un vector no nulo del retículo.*

Demostración. Sea $S' = \frac{1}{2}S$, así que $Vol(S') > det(\mathcal{L})$.
Entonces existen $x, y \in S'$, con $x \neq y \;:\; x - y \in \mathcal{L}$ tal que para algunos $v_1, v_2 \in \mathcal{L}, v_1 \neq v_2$:

$$v_1 + S' \cap (v_2 + S') \neq \varnothing$$

entonces,

$$z \;=\; v_1 + x \;=\; v_2 + y, \qquad x, y \in S'$$
$$x - y \;=\; v_2 - v_1 \;\neq\; 0, \qquad x, y \in S'$$

Ahora, $2x, 2y \in S$ por la definición de S', así que:

$$x - y \;= \frac{1}{2}(2x) + \frac{1}{2}(-2y) \in S$$

por la convexidad de S.

■

Como consecuencia se deduce el siguiente corolario:

Corolario 2.1.1 *Dado un retículo de rango máximo $\mathcal{L} \subset \mathbb{R}^n$:*

$$\lambda_1(\mathcal{L}) \;\leq\; \sqrt{n} \cdot (det(\mathcal{L}))^{\frac{1}{n}}$$

Demostración. Sea $B(0, r),\ \ r > \sqrt{n} \cdot det(\mathcal{L})^{\frac{1}{n}}$, convexo y simétrico.
Sea A un cubo de longitud $2 \cdot det(\mathcal{L})^{\frac{1}{n}}$. Entonces $A \subset B(0, r)$, donde:

$$d((1, 1, 1, \overset{n}{\ldots}, 1), (0, 0, 0, \overset{n}{\ldots}, 0)) = \sqrt{n}$$

Se sigue que:

$$vol(B(0, \sqrt{n} \cdot det(\mathcal{L})^{\frac{1}{n}})) \;\;>\;\; 2^n \cdot det(\mathcal{L})$$

Utilizando el teorema anterior, se ha probado que para una bola $B(0, r)$ con un radio $r > \sqrt{n} \cdot det(\mathcal{L})^{\frac{1}{n}}$ existe al menos un vector no nulo del retículo dentro de la misma. Por lo que el vector más corto del retículo tiene que ser menor o igual que el radio de la bola.

■

Nota 2.1.1 *Se pueden resumir los anteriores resultados en la siguiente expresión:*

$$\min_{i \in \{1,2,\ldots,n\}} \| b_i^* \| \;<\; \lambda_1(\mathcal{L}) \;<\; \sqrt{n} \cdot det(\mathcal{L})^{\frac{1}{n}} \tag{2.2}$$

No obstante, que se puedan calcular estas cotas no significa que sean suficientemente buenas.

Ejemplo 7 *Considérese el retículo bidimensional $\mathcal{L} = \{(1, 0), (0, 10000)\}$. Entonces el vector más pequeño es de longitud 1, sin embargo, el determinante es: $det(\mathcal{L}) = 10000$ y la cota según los resultados anteriores sería:*

$$\sqrt{2} \cdot 10000^{\frac{1}{2}} = \sqrt{20000} \gg 1 \tag{2.3}$$

2.2. Problema del vector más corto

A continuación se define el problema del vector más corto, teniendo en cuenta lo siguiente. Un problema de decisión es aquel en el que la cuestión tiene una respuesta binaria: 'SÍ' o 'NO'. Los problemas de optimización son aquellos en las que se busca minimizar o maximizar el valor de una salida dentro de un grupo de salidas generadas para una entrada específica. Nótese que cualquier problema de optimización siempre puede ser manejado como uno de decisión, fijando de antemano un valor objetivo.

Definición 2.2.1 *Dada una base $B \in \mathbb{Z}^{m\times n}$ de un retículo $\mathcal{L}$, se define el problema del vector más corto (Shortest Vector Problem, SVP) de la siguiente manera:*

- *Búsqueda: Encontrar un vector $v \in \mathcal{L} \setminus \{0\}$ tal que $\| v \| = \lambda_1(\mathcal{L})$.*
- *Optimización: Encontrar $\lambda_1(\mathcal{L})$.*
- *Decisión: Dado un número racional $r \in \mathbb{Q}$, determinar si $\lambda_1(\mathcal{L}) \leq r$.*

A continuación se plantea una variante del problema del vector más corto, que en particular puede resultar de interés cuando el valor objetivo es difícil de conseguir.

Definición 2.2.2 *Dada una base $B \in \mathbb{Z}^{m\times n}$ de un retículo $\mathcal{L}$, se habla del problema del vector más corto (en su variante de aproximación, SVP_γ), con $\gamma(n) > 1$ si:*

- *Búsqueda: Encontrar un vector $v \in \mathcal{L} \setminus \{0\}$ tal que $\| v \| = \gamma(n) \cdot \lambda_1(\mathcal{L})$.*
- *Optimización: Encontrar $z \in \mathbb{Q}$ tal que $z \leq \lambda_1(\mathcal{L}) \leq \gamma(n) \cdot z$.*
- *Promesa: Dado un número racional $r \in \mathbb{Q}$ y los siguientes conjuntos:*

$$SI = \{(B, r) : \lambda_1 \leq r\}$$

$$NO = \{(B, r) : \lambda_1 \geq \gamma(n) \cdot r\}$$

Este problema consiste en diferenciar acertadamente dada una instancia $I \in SI \cup NO$, a cuál pertenece I. La variante promesa se puede denotar de la siguiente manera: $GapSVP_\gamma$.

Algunas observaciones que se deducen a partir de la definición anterior:

- $SI \cap NO = \varnothing$.
- Para toda instancia I, $I \in SI \cup NO$.
- Si $I \notin SI \cup NO$, entonces I toma como válida cualquiera de las dos respuestas.

A raíz del problema del vector más corto nace el problema del vector más cercano. Para ello se define la distancia mínima como sigue:

$$dist(\mathcal{L}, t) = \min_{v\in\lambda\setminus\{0\}} \| v - t \| \tag{2.4}$$

2.2.1. Problema del vector más cercano

Definición 2.2.3 *Dada una base $B \in \mathbb{Z}^{m \times n}$ de un retículo $\mathcal{L}$ y $t \in \mathbb{Z}^m$, se habla del problema del vector más cercano (Closest Vector Problem, CVP) si:*

- *Búsqueda: Encontrar un vector $v \in \mathcal{L} \setminus \{0\}$ tal que $dist(\mathcal{L}, t) = \| v - t \|$ sea mínima.*
- *Optimización: Encontrar $dist(\mathcal{L}, t)$.*
- *Decisión: Dado un número racional $z \in \mathbb{Q}, z > 0$, decidir si existe un $v \in \mathcal{L} \setminus \{0\}$ tal que $dist(\mathcal{L}, t) \leq z$.*

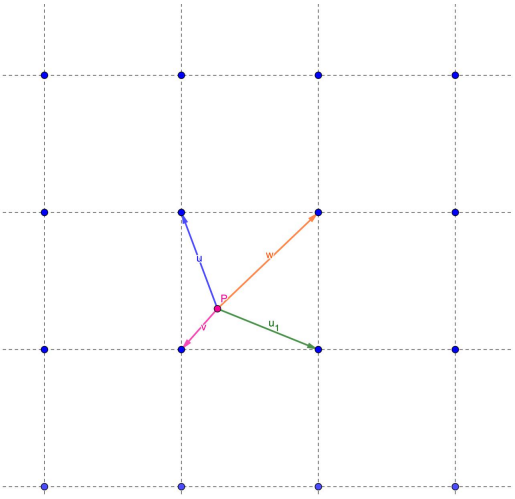


Figura 2.2. Representación del problema del vector más cercano en el plano euclídeo

También se considera una versión aproximada de este problema:

Definición 2.2.4 *Dada una base $B \in \mathbb{Z}^{m \times n}$ de un retículo $\mathcal{L}$ y $t \in \mathbb{Z}^m$, se define el problema del vector más cercano (en su variante de aproximación, CVP_γ), con $\gamma(n) > 1$ si:*

- *Búsqueda: Encontrar un vector $v \in \mathcal{L} \setminus \{0\}$ tal que $dist(\mathcal{L}, t) = \| v - t \| \leq \gamma(n) \cdot dist(\mathcal{L}, t)$.*
- *Optimización: Encontrar $z \in \mathbb{Q}$ tal que $z \leq dist(\mathcal{L}, t) \leq \gamma(n) \cdot z$.*
- *Promesa: Dado un número racional $x \in \mathbb{Q}$ y los siguientes conjuntos:*

$$SI = \{(B, t, x) : dist(\mathcal{L}, t) \leq x\}$$

$$NO = \{(B, t, x) : dist(\mathcal{L}, t) \geq \gamma(n) \cdot x\}$$

Como ya se mencionó anteriormente, este problema consiste en diferenciar de forma acertada dada una instancia $I \in SI \cup NO$, a cuál pertenece I. La variante promesa se puede denotar de la siguiente manera: $GapCVP_\gamma$.

2.2.2. Algoritmos de resolución

Sea $\mathcal{L}$ un retículo con una base dada $B \in \mathbb{Z}^{n\times n}$. Sea $w \in \mathcal{L}$. Entonces existen $\alpha_1, \alpha_2, \ldots, \alpha_n \in \mathbb{Z}$ tal que:

$$\| w \|^2 \;=\; \| \alpha_1 v_1 + \alpha_2 v_2 + \ldots + \alpha_n v_n \|^2$$

Aplicando las propiedades de la norma y el producto interno se tiene que:

$$\| w \|^2 \;= < \alpha_1 v_1 + \alpha_2 v_2 + \ldots + \alpha_n v_n \;,\; \alpha_1 v_1 + \alpha_2 v_2 + \ldots + \alpha_n v_n > \;=$$
$$= \alpha_1\alpha_1 < v_1, v_1 > + \ldots \alpha_n\alpha_n < v_n, v_n > \;=\; {\alpha_1}^2 \| v_1 \| + \ldots {\alpha_n}^2 \| v_n \|$$

Por tanto,

$$\| w \|^2 \;=\; \sum_{i=1}^{n} \alpha_i^2 \| v_i \|^2 \tag{2.5}$$

Como los coeficientes deben ser enteros, entonces los vectores más cortos de $\mathcal{L}$ deben ser los más cortos de $\{\pm v_1, \pm v_2, \ldots, \pm v_n\}$.

Ahora se analiza el caso del vector más cercano. Sea $x \in \mathbb{R}^n$. Entonces:

$$x = \beta_1 v_1 + \ldots + \beta_n v_n, \qquad \textit{con } b_i \in \mathbb{R},\ \forall i = 1, \ldots, n$$

Sea $w \in \mathcal{L}$, con $w = \alpha_1 v_1 + \alpha_2 v_2 + \ldots + \alpha_n v_n$, siendo $\alpha_i \in \mathbb{Z}, \forall i = 1, \ldots, n$. Entonces:

$$\| w - x \|^2 \;=\; \| (\alpha_1 - \beta_1) v_1 + (\alpha_2 - \beta_2) v_2 + \ldots + (\alpha_n - \beta_n) v_n \|^2 \;=$$
$$= < (\alpha_1 - \beta_1) v_1 + \ldots + (\alpha_n - \beta_n) v_n \;,\; (\alpha_1 - \beta_1) v_1 + \ldots + (\alpha_n - \beta_n) v_n > =$$
$$= \; (\alpha_1 - \beta_1)^2 \| v_1 \|^2 + \ldots + (\alpha_n - \beta_n)^2 \| v_n \|^2$$

Luego,

$$\| w - x \|^2 \;=\; \sum_{i=1}^{n} (\alpha_i - \beta_i) \| v_i \|^2 \tag{2.6}$$

Se tiene que $\alpha_i \in \mathbb{Z}$ y $b_i \in \mathbb{R}, \forall i = 1, \ldots, n$. Se puede minimizar la expresión tomando $\alpha_i = [\beta_i]$, siendo $[\beta_i]$ la parte entera de $\beta_i, \forall i = 1, \ldots, n$.

Como conclusión se deduce que cuánto más cortos y más ortogonales son los vectores de la base, mejores son las soluciones a la hora de resolver SVP y CVP. Si se piensa por el contrarrecíproco, es decir, si se quiere complicar el problema, cuánto mayor sea la base, más largos y menos ortogonales sean nuestros vectores, mayor es la dificultad de la resolución.

2.3. El problema de aprendizaje con errores

Considérese un vector secreto de coeficientes enteros $s = (s_1, s_2, \ldots, s_n) \in \mathbb{Z}^n$, y un sistema lineal de m ecuaciones de coeficientes conocidos cuya solución es s (con $m \geq n$), entonces se tiene que:

$$\begin{cases} a = a_{1,1}s_1 + a_{1,2}s_2 + \ldots + a_{1,n}s_n \\ b = a_{2,1}s_1 + a_{2,2}s_2 + \ldots + a_{2,n}s_n \\ \vdots \\ m = a_{m,1}s_1 + a_{m,2}s_2 + \ldots + a_{m,n}s_n \end{cases} \tag{2.7}$$

Resolver este sistema es relativamente sencillo aplicando cualquier método adecuado para esta tarea, como puede ser la *eliminación Gaussiana*. Sin embargo, eso varía cambiando ligeramente la situación. Concretamente se puede introducir un pequeño error en cada ecuación, dejándolas de la forma siguiente:

$$\begin{cases} a \approx a_{1,1}s_1 + a_{1,2}s_2 + \ldots + a_{1,n}s_n \\ b \approx a_{2,1}s_1 + a_{2,2}s_2 + \ldots + a_{2,n}s_n \\ \vdots \\ m \approx a_{m,1}s_1 + a_{m,2}s_2 + \ldots + a_{m,n}s_n \end{cases} \tag{2.8}$$

Resolver este problema no es tan simple, pues a la hora de aplicar la reducción por filas se acumulan errores de forma que el resultado final estaría posiblemente muy lejos del valor real.

Ejemplo 8 *En base a [8], considérese que existe un vector secreto $s = (2, 0)$, y los errores $e_1 = e_2 = -1$, entonces el sistema de ecuaciones con errores queda:*

$$\begin{cases} 2s_1 + 3s_2 \approx 5 \\ 3s_1 - s_2 \approx 7 \end{cases}$$

Si en este sistema se tuvieran igualdades, la solución sería $s = (2'36, 0'09)$ pero como se puede comprobar, ese resultado es diferente del vector secreto inicial $(2, 0)$. En la Fig. 2.3 se puede ver gráficamente la solución sin alterar y la solución con errores.

2.3.1. Formalización del problema

Sean $n \in \mathbb{N}$ y $q \in \mathbb{Z}$ (cuyo tamaño es similar a n).Considérese además m vectores $b_1, b_2, \ldots, b_m \in \mathbb{Z}_q^n$. El retículo Λ generado por la base de vectores $B = \{b_1, b_2, \ldots, b_m \in \mathbb{Z}_q^n\}$ sería por tanto el conjunto:

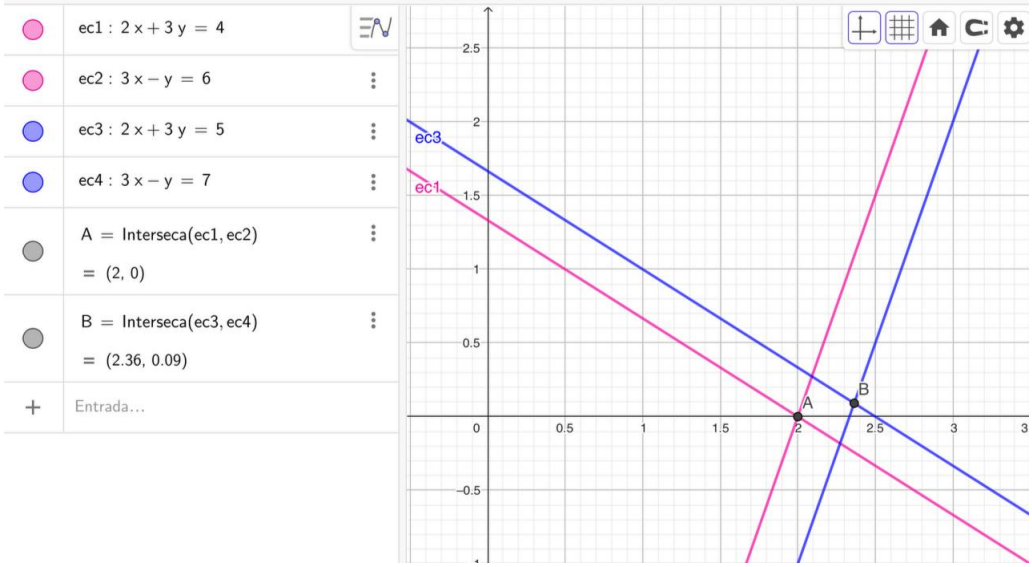


Figura 2.3. Representación gráfica del sistema de ecuaciones original (rosa) y el sistema alterado (azul)

$$\Lambda = \{\sum_{i=1}^{m} z_i \cdot b_i \ : \ z_i \in \mathbb{Z}\}$$

Fijada una distribución de probabilidad $\mathcal{X}$ sobre $\mathbb{Z}_q$, que de alguna forma permite elegir un cierto término de error de manera controlada. La selección de un error e de acuerdo con esta distribución se denota $e \leftarrow \mathcal{X}$. Estos parámetros sirven para fijar la llamada *distribución LWE* sobre $\mathbb{Z}_q^n \times \mathbb{Z}_q$.

Definición 2.3.1 *Distribución LWE. Sean $n, q \in \mathbb{N}$, $s \in \mathbb{Z}_q^n$ y $\mathcal{X}$ una distribución de probabilidad sobre $\mathbb{Z}_q$. Se dice que la distribución LWE módulo q asociada a s, denotada $\mathcal{A}_{s,\mathcal{Z}}$, es la definida eligiendo un vector $a \in \mathbb{Z}_q$ uniformemente al azar y eligiendo un error $e \leftarrow \mathcal{X}$, seleccionando así:*

$$(a, b) \in \mathbb{Z}_q^n \times \mathbb{Z}_q, \quad con \ b \ = < s, a > + \ e, \quad mod \ q \tag{2.9}$$

De forma natural se puede asociar al problema *LWE* un problema de búsqueda.

- Búsqueda *LWE*. Sean $n, m, q \in \mathbb{N}$ y sea s escogido uniformemente al azar en $\mathbb{Z}_q^n$. y además se considera $\mathcal{A}_{s,\mathcal{Z}}$ la distribución anteriormente definida. El problema consiste en calcular s dados m elementos de $\mathbb{Z}_q^n \times \mathbb{Z}_q$ elegidos de manera independiente.

2.3.2. Aprendizaje con errores sobre anillos

Según [9], siendo $n, m, q \in \mathbb{Z}$, el problema de aprendizaje con errores se basa en encontrar un vector $s \in \mathbb{Z}_q^n$ tal que $As + e = b \ mod \ q$, donde $A \in \mathbb{Z}_q^{m \times n}$,

$e, b \in \mathbb{Z}_q^n$. El vector e es el error y viene dado por la distribución $\mathcal{X}^m$ de probabilidad en $\mathbb{Z}_q^m$.

Se toma el anillo $\mathcal{K}_q = \mathbb{Z}_q[x]/(x^n + 1)$, con n siendo $2^{n'-1}$ tal que $(x^n + 1)$ es el $2^{n'}$ - ésimo polinomio ciclotómico. Entonces el par (A, b) viene dado por $A \in \mathcal{K}_q$ y $b = As + e \mod qK$ con $s \in \mathcal{K}_q$ y e obtenido de la distribución de probabilidad $\mathcal{X} \mod q$. Esto es el llamado *Module-LWE*.

La principal ventaja del esquema *Ring-LWE* es la eficiencia, en términos de velocidad, como del tamaño de la clave del texto cifrado. En cambio sus desventajas son la preocupación de que la estructura adicional pueda permitir ataques más eficientes y que las compensaciones entre eficiencia y seguridad solo se pueden escalar de forma bastante aproximada.

2.4. Complejidades computacionales asociadas

Se introducen una serie de resultados para entender las complejidades computacionales de cada problema. Para profundizar más se puede consultar [4].

Teorema 2.4.1 *El problema CVP en su versión de decisión es NP - completo.*

El siguiente resultado muestra que para $\gamma \geq 1$ encontrar soluciones a SVP_γ no es más difícil que encontrar soluciones a CVP_γ. Nótese que para $\gamma = 1$ resulta la versión primigenia.

Teorema 2.4.2 $\forall \gamma \geq 1$ *dado acceso a un oráculo que resuelva $GapCVP_\gamma$, es posible resolver $GapSVP_\gamma$ en tiempo polinomial.*

En pocas palabras, si no se resuelve en tiempo polinomial $GapSVP_\gamma$, tampoco $GapCVP$. Además, se debe aclarar que el término *oráculo* es una máquina abstracta usada para estudiar problemas de decisión y que hace referencia a la opacidad del proceso del cálculo del ejercicio en cuestión.
Para el caso aproximado se tiene que el siguiente resultado:

Teorema 2.4.3 *Para toda $c > 0$ y $\gamma(n) = n^{\frac{c}{\overline{log(log(n))}}}$. Entonces:*

$$GapCVP_\gamma \in NP - Difícil$$

En cuanto al problema del vector más corto, $GapSVP_\gamma$, se conjetura que no existen algoritmos clásicos (pre-cuánticos) que puedan lograr soluciones aproximadas con factores de aproximación lineales. Incluso, se considera la conjetura de que no existen algoritmos cuánticos que puedan resolver este problema con factores de aproximación lineales.

2.4.1. Glosario

En esta sección se repasan algunos términos relacionados con las complejidades computacionales.

Definición 2.4.1 *Sean $f, g : \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$ funciones. Entonces se llama notación asintótica a:*

- *$f(n) \in \mathcal{O}(g(n))$ si existe una constante $c \in \mathbb{R}^+$ y un $n_0 \in \mathbb{N}$ tales que:*

$$\forall n \geq n_0, \;\; f(n) \leq c \cdot g(n) \tag{2.10}$$

con $\mathcal{O}(g(n))$ cota superior asintótica.

$$\mathcal{O}(f(n)) \;\; = \;\; \{h : \mathbb{N} \to \mathbb{N} \;/\; \exists\, c > 0, n_0 > 0 : \;\; h(m) \leq c \cdot f(m), \;\; \forall m \geq n_0\}$$

Siendo $\mathcal{O}(f)$ el conjunto de las funciones acotadas por $c \cdot f$ para algún $c \in \mathbb{R}$.

En la teoría de la complejidad computacional, los problemas computacionales son modelados como subconjuntos $L \subset \{(0,1)^*\}$, llamados lenguajes, donde para todo $x \in \{(0,1)^*\}$ o bien $x \in L$ o bien $x \notin L$. Se introduce a continuación el tiempo polinomial.

Definición 2.4.2 *Un algoritmo $\mathcal{A}$ ejecuta en tiempo polinomial, si existe un polinomio $p : \mathbb{N} \longrightarrow \mathbb{N}$ tal que:*
$\forall x \in \{(0,1)^*\}$ el cómputo de $\mathcal{A}[x]$ termina a lo sumo en $p(|x|)$ pasos, donde $|x|$ denota el largo de la cadena de caracteres de x.

2.4.2. P vs NP

Sin entrar en profundidad, se sabe que hay distintas clases y las más importantes son: P, NP - Completo y NP - Difícil.

Definición 2.4.3 *La clase P se define como el conjunto de todos los problemas de decisión que pueden ser decididos por un algoritmo (determinista) en tiempo polinomial.*

La clase NP es el conjunto de los problemas que se pueden comprobar en tiempo polinomial.

Definición 2.4.4 *Un lenguaje de decisión $L \subset \{(0,1)^*\}$ está en NP si existe un polinomio $p : \mathbb{N} \longrightarrow \mathbb{N}$ y un algoritmo en tiempo polinomial $\mathcal{M}$ tal que para todo $x \in \{(0,1)^*\}$:*

$$x \in L \iff \exists\, u \in \{(0,1)^{p(|x|)}\} \;\; : \;\; \mathcal{M}(x,u) = 1 \tag{2.11}$$

Si $x \in L$ y $u \in \{(0,1)^*\}$ satisfacen que $\mathcal{M}(x,u) = 1$, se dice que u es un certificado para x.

Para poder tratar con cierto rigor sobre las clases NP-Difícil y NP-Completo se debe introducir la reducción en tiempo polinomial, ya que es un ingrediente clave en la definición de los mismos.

Definición 2.4.5 *Un lenguaje $L \subset \{(0,1)^*\}$ se reduce en tiempo polinomial al lenguaje $L' \subset \{(0,1)^*\}$, denotado $L \leq_p L'$, si existe una función computable en tiempo polinomial $f : \{(0,1)^*\} \longrightarrow \{(0,1)^*\}$ tal que:*

$$\forall x \in \{(0,1)^*\},\ x \in L \iff f(x) \in L' \tag{2.12}$$

Es decir, una reducción en tiempo polinomial se puede ver como una función que computa instancias que pertenecen a L a instancias que también pertenecen a L'. Como se tiene una doble implicación se puede manipular de igual manera con los contrarrecíprocos. La ventaja de esta definición es que si se tiene una función que transforma instancias de L en L' y además un algoritmo que resuelve L', también se tiene un algoritmo que resuelva L.

Definición 2.4.6 *Un lenguaje L' es NP-Difícil si para todo $L \in NP$, $L \leq_q L'$.*

Definición 2.4.7 *Un lenguaje L' es NP-Completo si es NP-Difícil y $L' \in NP$.*

Lo que estas definiciones quieren decir es que todo problema $L' \in$ NP-Difícil es tan complicado como cualquiera NP, esto es porque existe una reducción polinomial de L a L' por definición. Además, si se impone que $L' \in NP$ entonces se obtienen los problemas NP-Completos, que serían los más complicados de resolver.

Ahora que ya se ha dado una noción sobre las distintas clases computacionales se puede entender la relevancia del problema $P\ vs\ NP$. Es lógico ver que todo problema que se resuelve fácilmente se comprueba con un esfuerzo parecido, por lo que es claro que $P \subset PN$, sin embargo, $\quad ¿NP \subset P?$ En el caso de que $P = NP$ esto supondría que para cualquier problema existe un algoritmo que resuelve de manera eficaz el ejercicio en cuestión, sin necesidad de emplear la fuerza bruta.

Este dilema es uno de los 7 problemas del milenio. La comunidad científica cree y avala que se avanza en dirección a que $P \neq NP$, y es por eso que se escogen los problemas SVP, CVP y LWE como base de algunos esquemas (como es el nuestro en este caso: CRYSTALS-Dilithium), debido a que es realmente difícil resolverlos en un tiempo razonable, ya sea usando un ordenador clásico como uno cuántico.

3

CRYSTALS-Dilithium

En este capítulo se incluye una introducción al esquema de firma CRYSTALS - Dilithium basado en criptografía de clave pública. La dificultad de este esquema de firma digital se basa en el problema de encontrar los vectores más cortos en un retículo y en el ya mencionado *LWE*.

La criptografía de clave pública (también llamada asimétrica) emplea un par de claves (pública y privada) para definir los distintos roles, como pueden ser emisor/receptor, firmante/verificador, etc. de las dos entidades involucradas en el proceso. Se basa en problemas computacionales que son eficientes de realizar en un sentido, pero que no es factible realizarlos en sentido contrario (unidireccionalidad).

Definición 3.0.1 *Un esquema de firma de clave pública Sig es una terna de algoritmos ejecutables en tiempo polinomial $Sig = (\mathcal{K}, \mathcal{F}, \mathcal{V})$, donde:*

- *$\mathcal{K}$ es el algoritmo de generación de claves, es un algoritmo probabilístico que, recibiendo como entrada al parámetro de seguridad $l \in \mathbb{N}$, produce como salida un par (pk, sk) de clave pública y de clave privada.*
- *$\mathcal{F}$ es el algoritmo de firma, es un algoritmo probabilístico que, recibiendo como mensaje de entrada $\mathcal{M}$ codificado como una cadena de $p(l)$ bits, para un cierto polinomio p junto con la clave secreta sk, da como salida el mensaje M y su firma σ que, de nuevo, se codifica como una cadena de bits de longitud polinomial l.*
- *$\mathcal{V}$ es el algoritmo de verificación, es un algoritmo determinista que, recibiendo como entrada un par (M, σ) y una clave pública pk, da como salida un bit que puede ser 0 o 1 indicando el fracaso o el éxito de la operación.*

3.1. Esquema inicial

A continuación se presentan dos esquemas. El primero de ellos es una versión simplificada denominada *Template*, que se utiliza con el objetivo de facilitar la comprensión de la versión final. Para ello, se desarrollan algunos conceptos

fundamentales relacionados con la fase de firma digital.

Funciones *hash*. Una función *hash* es un algoritmo matemático que transforma cualquier dato entrante en una serie de caracteres de salida, con una longitud fija o variable, dependiendo del algoritmo *hash* que se esté utilizando. CRYSTALS-Dilithium utiliza alguna de las implementaciones de SHA-3 *(Secure Hash Algorithm 3)*, que se estandarizó en el 2015 por el NIST.
Se han generado algunos ejemplos utilizando SHAKE128. Los ejemplos siguientes han sido desarrollados con la ayuda de [10].

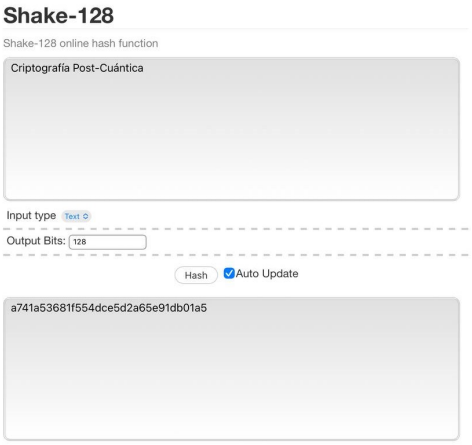


Figura 3.1.

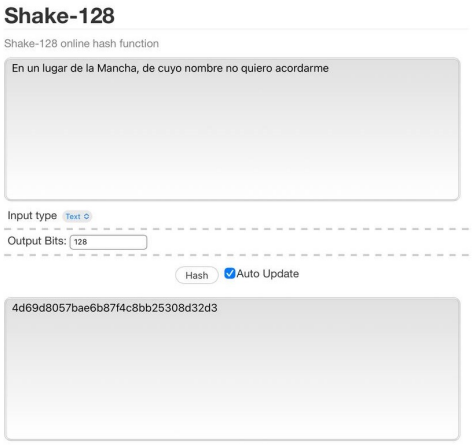


Figura 3.2.

El conjunto B_τ está implicado en la proceso de crear la firma. B_τ es el conjunto de elementos de $\mathcal{R}_q$ que tienen τ coeficientes que únicamente son 1, -1 y el resto son 0. Véase en la Fig. 3.3 el algoritmo que los genera de forma aleatoria.

SampleInBall(ρ)
01 Initialize $\mathbf{c} = c_0 c_1 \dots c_{255} = 00 \dots 0$
02 **for** $i := 256 - \tau$ to 255
03 $\quad j \leftarrow \{0, 1, \dots, i\}$
04 $\quad s \leftarrow \{0, 1\}$
05 $\quad c_i := c_j$
06 $\quad c_j := (-1)^s$
07 **return c**

Figura 3.3. Algoritmo que calcula un elemento de B_τ de forma aleatoria

Se expone un resultado para caracterizar los conjuntos cocientes, teniendo anillos e ideales de polinomios.

Proposición 3.1.1 *Sean A un cuerpo e I un ideal de A, siendo $I = p(x)$ con $p(x) \in A[x]$ un polinomio de grado n. Entonces se tiene que:*

$$A/I \ = \ \{(a_0 + a_1x + a_2x^2 + \dots a_{n-1}x^{n-1}) + I : a_i \in A\} \tag{3.1}$$

Ejemplo 9 *Sean $\tau = 2$ y el anillo $\mathbb{Z}_3[x]/(x^4+1)$, por Ec. (3.1) se tiene que:*

$$\mathbb{Z}_7[x]/(x^3+1) = \{(a_0 + a_1x + a_2x^2) + I/a_i \in \{0, 1, \dots, 6\}, \forall i\}$$

Por tanto, en este caso:

$$B_\tau = \{(a_0 + a_1x + a_2x^2) + I/a_i \in \{0, 1, -1\}$$

Elementos de B_τ podrían ser:

- $1 + I$.
- $(1 + x) + I$.
- $(-1 - x^2) + I = (6 + 6x^2) + I$. *Pues se considera $\mathbb{Z}_7$ y $-1 = 6$.*

Reducciones modulares. Para cualquier número entero par positivo α se define $r' = r \ mod^{\pm} \ \alpha$, con r' el único $r \in (-\frac{\alpha}{2}, \frac{\alpha}{2}]$ tal que $r' \equiv r \ mod \ \alpha$. En el caso de α impar es similar, únicamente varía en el intervalo dónde se encuentra r, siendo $[-\frac{\alpha-1}{2}, \frac{\alpha-1}{2}]$. Además, para cualquier entero positivo α se define $r' = r \ mod^{+} \ \alpha$, siendo r' el único elemento $r' \in [0, \alpha)$ tal que $r' \equiv r \ mod \ \alpha$.

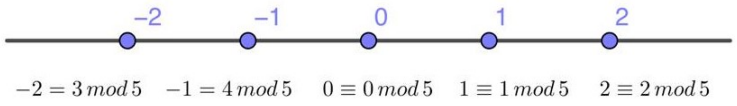


Figura 3.4. Representación gráfica de una reducción modular con $\alpha = 5$

En las Fig. 3.5 y 3.6 se pueden apreciar los algoritmos complementarios y la versión inicial del esquema de firma. Ahora que ya se han introducido los conceptos necesarios, se procede a estudiar cada algoritmo del *Template*.

Como en cualquier esquema de firma, el esquema completo de CRYSTALS-Dilithium consta de tres algoritmos: Generación de clave, proceso de firma y proceso de verificación.

$\underline{\mathsf{Power2Round}_q(r, d)}$
08 $r := r \bmod^+ q$
09 $r_0 := r \bmod^\pm 2^d$
10 **return** $\big((r - r_0)/2^d, r_0\big)$

$\underline{\mathsf{MakeHint}_q(z, r, \alpha)}$
11 $r_1 := \mathsf{HighBits}_q(r, \alpha)$
12 $v_1 := \mathsf{HighBits}_q(r + z, \alpha)$
13 **return** $[\![r_1 \neq v_1]\!]$

$\underline{\mathsf{UseHint}_q(h, r, \alpha)}$
14 $m := (q - 1)/\alpha$
15 $(r_1, r_0) := \mathsf{Decompose}_q(r, \alpha)$
16 **if** $h = 1$ and $r_0 > 0$ **return** $(r_1 + 1) \bmod^+ m$
17 **if** $h = 1$ and $r_0 \leq 0$ **return** $(r_1 - 1) \bmod^+ m$
18 **return** r_1

$\underline{\mathsf{Decompose}_q(r, \alpha)}$
19 $r := r \bmod^+ q$
20 $r_0 := r \bmod^\pm \alpha$
21 **if** $r - r_0 = q - 1$
22 **then** $r_1 := 0$; $r_0 := r_0 - 1$
23 **else** $r_1 := (r - r_0)/\alpha$
24 **return** (r_1, r_0)

$\underline{\mathsf{HighBits}_q(r, \alpha)}$
25 $(r_1, r_0) := \mathsf{Decompose}_q(r, \alpha)$
26 **return** r_1

$\underline{\mathsf{LowBits}_q(r, \alpha)}$
27 $(r_1, r_0) := \mathsf{Decompose}_q(r, \alpha)$
28 **return** r_0

Figura 3.5. Algoritmos complementarios al esquema CRYSTALS-Dilithium

$\underline{\mathsf{Gen}}$
01 $\mathbf{A} \leftarrow R_q^{k \times \ell}$
02 $(\mathbf{s}_1, \mathbf{s}_2) \leftarrow S_\eta^\ell \times S_\eta^k$
03 $\mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$
04 **return** $(pk = (\mathbf{A}, \mathbf{t}), sk = (\mathbf{A}, \mathbf{t}, \mathbf{s}_1, \mathbf{s}_2))$

$\underline{\mathsf{Sign}(sk, M)}$
05 $\mathbf{z} := \perp$
06 **while** $\mathbf{z} = \perp$ do
07 $\mathbf{y} \leftarrow S_{\gamma_1 - 1}^\ell$
08 $\mathbf{w}_1 := \mathsf{HighBits}(\mathbf{A}\mathbf{y}, 2\gamma_2)$
09 $c \in B_\tau := \mathrm{H}(M \parallel \mathbf{w}_1)$
10 $\mathbf{z} := \mathbf{y} + c\mathbf{s}_1$
11 **if** $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or $\|\mathsf{LowBits}(\mathbf{A}\mathbf{y} - c\mathbf{s}_2, 2\gamma_2)\|_\infty \geq \gamma_2 - \beta$, then $\mathbf{z} := \perp$
12 **return** $\sigma = (\mathbf{z}, c)$

$\underline{\mathsf{Verify}(pk, M, \sigma = (\mathbf{z}, c))}$
13 $\mathbf{w}_1' := \mathsf{HighBits}(\mathbf{A}\mathbf{z} - c\mathbf{t}, 2\gamma_2)$
14 **if return** $[\![\|\mathbf{z}\|_\infty < \gamma_1 - \beta]\!]$ **and** $[\![c = \mathrm{H}(M \parallel \mathbf{w}_1')]\!]$

Figura 3.6. *Template*, Primera versión de CRYSTALS-Dilithium

- Generación de clave. El algoritmo empieza generando una matriz aleatoria de dimensión $k \times l$, en particular se suele tomar 5×4 o 6×5. Los elementos de esta matriz pertenecen a $\mathcal{R}_q$. Se sabe que $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$, donde $q = 2^{23} - 2^{13} + 1$ y $n = 256$. Dado que q es un número primo, se tiene que $\mathbb{Z}_q$ es un cuerpo, por lo que esto implica que $\mathbb{Z}_q[x]$ también lo es. Se deduce de Eq. 3.1 que los elementos de $\mathcal{R}_q$ denotando como $I = (x^{256} + 1)$ se expresan de la siguiente forma:

$$\mathcal{R}_q \ = \ \{(a_0 + a_1 x + a_2 x^2 + \ldots + a_{255} x^{255} \ + \ I : a_i \in \mathbb{Z}_q\}$$

  En el segundo paso se generan dos vectores s_1 y s_2, siendo $(s_1, s_2) \in S_\mu^l \times S_\mu^k$ y S_μ el conjunto de todos los elementos de $\mathcal{R}_q$ tal que $\| w \|_\infty \leq \mu$.
  Luego se computa $\mathbf{t} := As_1 + s_2$. En esta fase se aprecia cómo se relaciona con el problema de L.W.E. siendo s_1 el 'vector secreto' y s_2 el 'vector error'. Para finalizar este algoritmo se devuelve la clave tanto pública $pk = (A, t)$ como privada $sk = (A, t, s_1, s_2)$.

- Proceso de firma. Para iniciar este algoritmo son necesarios dos parámetros: el mensaje $\mathcal{M}$ y la clave secreta generada anteriormente.

  El proceso comienza con un bucle *while* que sirve para crear la firma digital. Se obtiene un vector $y \in S_{\gamma_1 - 1}^l$ de forma aleatoria, donde γ_1 se escoge arbitrariamente de manera que no sea ni lo suficientemente largo ni tampoco lo suficientemente corto, para que no sea fácil forzarla ni tampoco la firma puntual revele la clave.

  En el siguiente paso se define w_1, que es el resultado de aplicar HighBits a Ay y $2\gamma_2$. Nótese que γ_2 se obtiene de expresar todo elemento w del producto Ay como:

$$w = w_1 \cdot 2\gamma_2 + w_0, \qquad |w_0| \leq \gamma_2 \tag{3.2}$$

  A continuación se define un vector c, donde $c \in B_\tau$, que es el resultado de aplicar el algoritmo *hash*t al mensaje $\mathcal{M}$ con una longitud fija w_1. Se computa el candidato a firma $z = y + c \cdot s_1$, recordando que $s_1 \in sk$.

  Para terminar este proceso la firma z tiene que cumplir dos condiciones:

  1. $\| z \|_\infty \geq \gamma_1 - \beta$, siendo β el coeficiente más grande del producto $c \cdot s_i$, con $i = 1, 2$.
  2. $\| LowBits(Ay - c \cdot s_2, 2\gamma_2) \|_\infty \geq \gamma_2 - \beta$.

En el caso de que no se cumpla ninguna de las dos condiciones se repite el proceso. Finalmente una vez concluye este bucle *while*, se obtiene la firma digital:

$$\sigma = (z, c)$$

- Proceso de verificación. Para ejecutar este algoritmo se necesita como parámetro de entrada la clave secreta sk, el mensaje $\mathcal{M}$ y la firma $\sigma = (z, c)$. Bastará con definir $w_2 = HighBits(Az - ct, 2\gamma_2)$, y luego comprobar las condiciones pertinentes. En caso de que se cumplan ambas se nos devolverá un 1, y en caso contrario un 0.

Nota 3.1.1 *En sí, el esquema planteado es un tanto ineficiente, debido a que la matriz generada es una matriz de polinomios de tamaño $k \times l$. Es decir, se operan $k \cdot l$ polinomios, añadiendo que cada polinomio puede tener a lo sumo* 256 *coeficientes y además se encuentran en el intervalo $[0, q)$ con $q = 2^{23} - 2^{13} + 1 = 8380417$, por lo que se tiene que la matriz que genera la clave es de unas dimensiones muy considerables.*

3.2. Esquema final

A continuación se presenta la versión final de CRYSTALS-Dilithium, prestando especial atención a algunos aspectos que destacan con respecto al *Template*.

En la Fig. 3.7 se pueden ver algunas diferencias con respecto al *Template*, como pueden ser:

1. El uso de semillas para iniciar los procesos aleatorios, como puede ser los vectores y la matriz en el primer algoritmo.
2. Se utiliza la Transformación Teórica de Números, para optimizar gran parte de los cálculos que están involucrados en la generación aleatoria de los parámetros.
3. Se utilizan nuevos algoritmos (que ya han sido introducidos en la Fig. 3.5) tales como $Decompose_q$, $MakeHint_q$ o $Power2Round_q$.

En las próximas líneas, se exponen en mayor profundidad algunos de los procesos y algoritmos que se emplean en el esquema en cuestión.

- Transformación Teórica de Números, (NTT). En primer lugar, en base a [2] el polinomio $p(x) = x^n + 1$ se divide en $x - r^i \; mod \, q$, con $i = 1, 3, 5, \ldots, 511$ y $r = 1753$. Se sabe que:

$$\mathbf{Z}_q[x]/(x - r^i) \;\cong\; \mathbf{Z}_q \tag{3.3}$$

Como se ha comentado, $(x^n + 1) = (x - r) \cdot (x - r^3) \cdot \ldots (x - r^{511})$ y por tanto, por el Teorema Chino del Resto se tiene que:

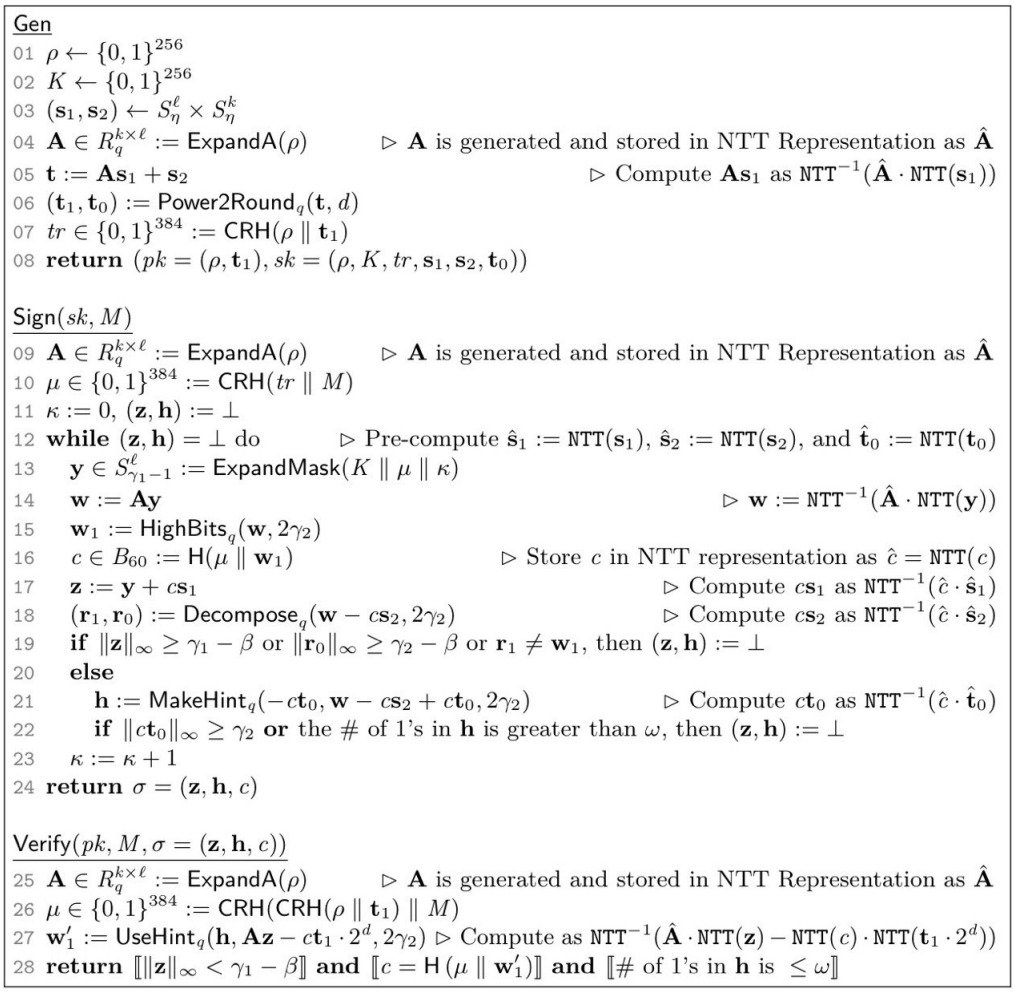


Figura 3.7. Última versión de CRYSTALS-Dilithium

$$\mathcal{R}_q \;=\; \mathbf{Z}_q[x]/(x^n+1) \;\cong\; \prod_{i=1} \mathbf{Z}_q[x]/(x-r^i) \;\cong\; \mathbf{Z}_q^{256} \tag{3.4}$$

Por tanto, se define la función NTT:

$$\begin{aligned} NTT\colon \mathcal{R}_q &\longrightarrow \mathbf{Z}_q^{256} \\ a &\longmapsto NTT(a) = \hat{a} \end{aligned}$$

dónde $\hat{a} = (a(r_0), a(-r_0), \ldots, a(r_{127}), a(-r_{127})) \in Z_q^{256}$ y además $r_i = r^{brv(128+i)}$ siendo la función $brv(n)$ la que invierte el número de bits de un entero n de 8 bits.

Ejemplo 10 *Se pretende observar el cálculo de* $a(r_i)$*. Para este propósito, se considera el valor de* $i = 1$*. En consecuencia:*

$$r_1 = r^{brv(128+1)} = r^{brv(129)}$$

Pero, 129 *en bits es* 10000001*, sin embargo si se aplica brv resulta que* $brv(10000001) = 10000001$*, ya que si se le da la vuelta queda el mismo número, pues es palíndromo. Por lo tanto:*

$$r_1 = r^{10000001},$$

con $r = 1753$ *y todas las operaciones mod q.*

- Expansión de la matriz. La función *ExpandA(p)* asigna uniformemente una semilla $p \in \{0,1\}^{256}$ a una matriz $A \in \mathcal{R}_q^{k \times l}$ en la representación NTT. Esto es debido a que la matriz solo es necesaria para la multiplicación y dadas sus dimensiones en aras de ahorrar coste computacional se aplica la Transformación Teórica de Números. Por lo que si se tiene $A \in \mathcal{R}_q^{k \times l}$ como entrada, el valor de salida es $\hat{A} \in \mathbf{Z}_q^{256}$.

- Muestreo de vectores. Con la función *ExpandS* se generan los vectores secretos en la fase de generación de clase, asignando una semilla p' a $(s_1, s_2) \in S_\mu^l \times S_\mu^k$. También se puede hablar de *ExpandMask* que se utiliza para generar de manera determinista la aleatoriedad del esquema de firma.

- Resistencia a las colisiones. En el esquema de firma Dilithium se utiliza una función *hash* resistente a las colisiones que se asigna a $\{0,1\}^{384}$. Estas funciones tienen la propiedad de que es difícil encontrar dos entradas que tengan la misma salida. Si se tiene una función *hash* con más entradas que salidas necesariamente habrá colisiones.

4

Implementaciones

Para la elaboración de este capítulo, se ha seleccionado la plataforma *Kaggle* [11] para implementar CRYSTALS-Dilithium. El objetivo principal ha sido desarrollar un código con fines didácticos. Por esta razón, se ha optado por el lenguaje de programación Python, ya que es conocido por ser un lenguaje de alto nivel y fácil de aprender. Además, Python ofrece una amplia variedad de módulos y bibliotecas que son útiles para el desarrollo de nuestro código. Para ver el código completo se puede consultar [12].
La implementación comienza importando las librerías necesarias:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load


import sympy, random, string, hashlib, sys
import numpy as np

from sympy import *
from math import *
```

Figura 4.1. Importación de las librerías Python

4.1. Funciones complementarias

A continuación se fijan aquellos valores que serán iguales a lo largo de todo el proceso, como pueden ser:

- El mayor número que puede representar Python, definido como: MAX_INT. Más concretamente: MAX_INT = 9223372036854775807.
- $q = 2^{23} - 2^{13} + 1$ mencionado anteriormente en la sección 3.1.

- En aras de aportar una implementación más dinámica, en cada iteración del programa completo se escoge un n aleatorio comprendido entre $[5, 15]$. Se puede tomar cualquier intervalo siempre y cuando $n > 0$.
- Se elige de antemano $\gamma_1 = 104659$, ya que no es ni tan grande ni tan pequeño comparado con q, tal y como se explicó en el *Template*. Además, se fija $\gamma_2 = [\frac{q}{3}]$, pues de ese modo se argumenta Eq. 3.2.

```
##Variables que necesitaremos

MAX_INT = sys.maxsize
q = 2**23- 2**13 + 1
n = np.random.randint(5,15)
gamma1 = 104659
gamma2 = q//3
```

Figura 4.2. Parámetros fijos a lo largo de la implementación

El siguiente paso consiste en definir todas las funciones complementarias, especialmente aquellas mencionadas en la Fig. 3.5.

```
def modpm(r, alpha):

    a = int((alpha-1)/2)+1
    b = int(r/2+1)
    r1 = r%alpha

    if alpha%2 == 0:
        if not r1 in range(b):
            r1 -= alpha
    else:
        if not r1 in range(a):
            r1 -= alpha

    return r1
```

Figura 4.3. Función para aplicar la reducción modular

En este momento, se presentan las funciones *highbits*, *lowbits* y la función utilizada para calcular la norma infinito de una matriz. Cabe destacar que para obtener la norma infinito de una matriz, se debe encontrar el valor máximo de la suma de los elementos de cada fila de dicha matriz.

```python
def decompose(r, alpha, q):
    r = r%q
    r0 = modpm(r, alpha)

    if r - r0 == q - 1:
        r1 = 0
        r0 = r0 - 1
    else:
        r1 = int((r - r0)/alpha)

    return (r1, r0)
```

Figura 4.4. Función para aplicar la descomposición módulo q

```python
def highbits(r, alpha, q):
    return decompose(r, alpha, q)[0]
```

```python
def lowbits(r, alpha, q):
    return decompose(r, alpha, q)[1]
```

```python
def inf(matriz):
    return max(np.linalg.norm(matriz, ord=np.inf, axis=1))
```

Figura 4.5. Funciones utilizadas en el proceso de creación de firma

Ejemplo 11 *Con el fin de comprender con mayor claridad las funciones highbits y lowbits, se ha implementado el siguiente código, dado que estos conceptos pueden resultar abstractos en un principio.*

```python
hb = []
lb = []

for i in range(1, 5):
    for j in range(1,5):
        hb.append(highbits(i,j,q))
        lb.append(lowbits(i,j,q))

hb, lb
```

```
([1, 1, 0, 1, 2, 1, 1, 1, 3, 1, 1, 1, 4, 2, 1, 1],
 [0, -1, 1, -3, 0, 0, -1, -2, 0, 1, 0, -1, 0, 0, 1, 0])
```

Figura 4.6. Ejemplo de la función *highbits* y *lowbits*

Antes de pasar a la función de creación de claves se expone la función con la que se aplica el *hash* al mensaje que se quiere firmar. Se han realizado algunas modificaciones al código respecto al *Template*. Entre estas modificaciones entra acotar el valor del *hash* al intervalo $[0, 10^{w_1}]$.

```
def hashing(text, w_1):

    hash_obj = hashlib.sha256(text.encode('utf-8'))
    hash_result = int.from_bytes(hash_obj.digest(), byteorder='big') % (10 ** (w_1))

    return hash_result
```

Figura 4.7. Función para aplicar el *hash* al mensaje

4.2. Implementación de CRYSTALS-Dilithium

La función *keygen* es la encargada de generar las claves pública y privada en el esquema de CRYSTALS-Dilithium. En este proceso, se genera una matriz de enteros aleatorios en el intervalo [0,MAX_INT], la cual es sometida a un módulo q. Es importante destacar que, a diferencia del *Template*, en el que cada elemento de la matriz A era un polinomio, en este caso se utilizan enteros. Además, los vectores $s1$ y $s2$ también se someten a un módulo de un número primo bajo para evitar posibles problemas relacionados con la manipulación de números excesivamente grandes durante la compilación.

Ejemplo 12 *Para una matriz cuadrada A y $n = 5$:*

Ahora se contempla la función *sign* que es la que se utiliza para crear la firma digital. Al igual que con la función *keygen*, se ha adaptado la implementación y se han realizado algunos cambios:

- En lugar de emplear el producto de la matriz A y un vector seleccionado al azar y, se utilizará el valor máximo del vector resultante Ay, con el fin de eliminar algunas dimensiones y reducir la complejidad computacional.
- El cálculo del elemento $c \in B\tau$ presenta diferencias. En el capítulo anterior se mencionó que c es un vector que contiene exactamente τ elementos $1's$ o $-1's$. No obstante, con el fin de ofrecer una implementación más didáctica, se ha considerado a c como un escalar.
- Al concluir la ejecución del algoritmo, se retorna no solo la firma $\sigma = (z, c)$, sino también el parámetro β. Esto se debe a que el siguiente algoritmo requiere de este valor para poder verificar la firma correctamente. De no contar con

```
                ###### Creación de las claves #######

def keygen(n, m, q):
    A = np.random.randint(MAX_INT, size=(n, m))
    A = A%q

    #Aplicamos modulo número primo arbitrario para simplificar los datos
    s1 = (np.random.randint(MAX_INT, size=(m, 1)))%119
    s2 = (np.random.randint(MAX_INT, size=(n, 1)))%119

    t = (np.matmul(A, s1) + s2)%q

    pk = (t, A)
    sk = (t, A, s1, s2)

    return pk, sk
```

Figura 4.8. Generación de las claves públicas y privadas

```
pk, sk = keygen(n,n,q)
pk, sk
```

```
((array([[1045211],
         [ 447971],
         [3320430],
         [1075765],
         [5108209]]),
  array([[4159266, 2419033,  932949, 2275241,  623961],
         [1249022, 8371351, 6632995, 6943278, 4043099],
         [7889044, 6643355, 5210322, 5938889, 4250367],
         [7867777, 4800784, 1712219, 4006432, 7672247],
         [3517696, 3348984, 6490516, 5363988, 6773847]])),
 (array([[1045211],
         [ 447971],
         [3320430],
         [1075765],
         [5108209]]),
  array([[4159266, 2419033,  932949, 2275241,  623961],
         [1249022, 8371351, 6632995, 6943278, 4043099],
         [7889044, 6643355, 5210322, 5938889, 4250367],
         [7867777, 4800784, 1712219, 4006432, 7672247],
         [3517696, 3348984, 6490516, 5363988, 6773847]]),
  array([[41],
         [41],
         [70],
         [50],
         [48]]),
  array([[113],
         [ 15],
         [ 23],
         [ 26],
         [ 39]])))
```

Figura 4.9. Ejemplo del primer algoritmo del esquema de firma

este parámetro y los datos correspondientes, no sería posible realizar dicha verificación.

```
                        ##### Creación de firma ######

def sign(sk, M):

    t = sk[0]
    A = sk[1]
    s1 = sk[2]
    s2 = sk[3]

    [n,m] = A.shape

    y = (np.random.randint(MAX_INT, size=(n, 1)))%(gamma1-1)
    Ay = int(max(np.matmul(A, y)))

    w_1 = highbits(Ay, 2*gamma2, q)
    c = hashing(M, abs(w_1))
    z = y + c*s1

    sigma = (z, c)

    ##Calculamos beta para la parte final, ya que necesitamos los máximos

    a = int(inf(c*s1))
    b = int(inf(c*s2))

    beta = int(max(a,b))

    if (inf(z) >= (gamma1 - beta)) or (lowbits(Ay - b , 2*gamma2, q) >= (gamma2 - beta)):
        sigma, beta = sign(sk, M)

    return sigma, beta
```

Figura 4.10. Función para firmar el mensaje

Ejemplo 13 *Continuando con los datos de la Fig. 4.9, se procede a realizar un ejemplo sobre el proceso de creación de la firma.*

Para finalizar con estas funciones principales, se presenta la función con la que se verifica la firma. En este algoritmo a excepción de introducir el parámetro β no se altera el código con respecto al *Template*.

Ejemplo 14 *Continuando con la Fig. 4.11 se presenta un caso en el cual es posible verificar el mensaje correctamente. Posteriormente, se expone otro caso en el que se obtiene una respuesta negativa al intentar realizar la verificación.*

```
sigma, beta = sign(sk, "Mensaje a firmar")
sigma, beta
```

```
((array([[ 61032],
         [ 37154],
         [ 92029],
         [  6904],
         [101487]]),
  5),
 565)
```

Figura 4.11. Ejemplo del segundo algoritmo de CRYSTALS-Dilithium

```
                    ######## Verificación de firma #########

def verify(pk, M, sigma, beta):

    t = pk[0]
    A = pk[1]

    z = sigma[0]
    c = sigma[1]

    Az = inf(np.matmul(A, z))
    ct = inf(c*t)

    w1 = highbits(Az - ct, 2*gamma2, q)

    return inf(z) < (gamma1 - beta) and c == hashing(M, w1)
```

Figura 4.12. Función para verificar la firma

```
verify(pk, "Mensaje a firmar", sigma, beta)
```

```
True
```

```
verify(pk, "Mensaje", sigma, beta)
```

```
False
```

Figura 4.13. Ejemplo del último algoritmo de CRYSTALS-Dilithium

4.3. Experimentos

Finalmente, se debe asegurar empíricamente el correcto funcionamiento de los algoritmos. Con este fin, se han definido dos funciones complementarias: la primera genera cadenas aleatorias a partir de una entrada que se toma como la longitud de las mismas; mientras que la segunda función se encarga de llevar a cabo todo el proceso de creación de claves, firma y verificación de la misma. Esta última función proporciona una respuesta que indica si el proceso se realizó con éxito (*True*) o si fracasó (*False*).

```
def get_random_string(size):
    chars = string.ascii_lowercase+string.ascii_uppercase+string.digits
    return ''.join(random.choice(chars) for _ in range(size))
```

Figura 4.14. Función auxiliar para generar strings aleatorios

```
def run(message_len):
    pk, sk = keygen(n, n, q)
    message = get_random_string(message_len)
    #print(message)
    sigma, beta = sign(sk, message)
    return verify(pk, message, sigma, beta)
```

Figura 4.15. Función auxiliar para llevar a cabo los experimentos

En el primer experimento realizado (ver Fig. 4.16), se ha observado de manera dinámica cómo, al generar cadenas aleatorias de 20 caracteres, se ha llevado a cabo el proceso de verificación con éxito. En este experimento se han realizado diez iteraciones y se ha comprobado el correcto funcionamiento del algoritmo de verificación en todas ellas.

Por último, para obtener una prueba más confiable (ver Fig.4.17), se ha realizado el mismo proceso descrito anteriormente un total de 100000 veces. En cada iteración se ha generado un string aleatorio de longitud 20 y se ha inicializado un contador llamado *cnt* a cero. Por cada verificación que ha devuelto *False*, se ha actualizado *cnt* añadiéndole 1. Al final de las 100000 iteraciones, el valor de *cnt* ha sido igual a cero, lo que indica que todas las verificaciones han funcionado correctamente y que se puede afirmar que la implementación no presenta problemas de autenticación.

```python
### Primer experimento ###

cnt = 0

for i in range(10 ** 1):
    flag = run(20)
    if not flag:
        cnt += 1
    print('Iteration:', i + 1, '/ Errors:', cnt)
```

```
Iteration: 1 / Errors: 0
Iteration: 2 / Errors: 0
Iteration: 3 / Errors: 0
Iteration: 4 / Errors: 0
Iteration: 5 / Errors: 0
Iteration: 6 / Errors: 0
Iteration: 7 / Errors: 0
Iteration: 8 / Errors: 0
Iteration: 9 / Errors: 0
Iteration: 10 / Errors: 0
```

Figura 4.16. Primer experimento

```python
### Segundo experimento del esquema de firma ###

cnt = 0

for i in range(10 ** 5):
    flag = run(20)
    if not flag:
        cnt += 1
print(cnt)
```

```
0
```

Figura 4.17. Segundo experimento

5

Conclusiones

Este trabajo ha introducido algunos aspectos de la situación actual de la criptografía post-cuántica, centrándose en el esquema de firma digital CRYSTALS-Dilithium. Desde un punto de vista teórico y a la vez didáctico, se han introducido las bases del esquema analizado, definiendo sus cimientos matemáticos, que son los retículos, además del problema computacional sobre el que se basa y los algoritmos que componen el esquema CRYSTALS-Dilithium. Por último, se aporta una implementación del esquema de firma, donde también se comentan las pruebas que se han realizado para confirmar que el código no presenta problemas de autenticación.

Como trabajos futuros, se propone la exploración de la posibilidad de llevar a cabo un estudio teórico y la correspondiente implementación práctica de los principales esquemas de firma post-cuántica, como CRYSTALS-Dilithium, FALCON y SPHINCS+. Estos esquemas podrían ser comparados y evaluados en diferentes condiciones y escenarios para determinar cuál de ellos ofrece un mejor desempeño. Esa investigación permitiría obtener una visión más completa y precisa de las características, fortalezas y debilidades de cada esquema, y podría brindar información valiosa para el desarrollo y la elección de soluciones de firma digital post-cuántica en el futuro.

A

Artículo aceptado en congreso

A.1. VIII Jornadas Nacionales de Investigación en Ciberseguridad

Estudio del esquema de firma CRYSTALS-Dilithium.
Édgar Pérez Ramos, Pino Caballero-Gil.
VIII Jornadas Nacionales de Investigación en Ciberseguridad (JNIC).
Vigo, 21 a 23 de junio de 2023.

Bibliografía

[1] NIST, "NIST Announces First Four Quantum-Resistant Cryptographic Algorithms," *NIST News*, Jul. 2022. [Online]. Disponible en: https://tinyurl.com/NIST-news.

[2] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, y D. Stehlé, "CRYSTALS-Dilithium, Algorithm Specifications and Supporting Documentation (Version 3.1)", 2022. Disponible en: https://pq-CRYSTALS.org/dilithium/data/dilithium-specification-round3-20210208.pdf.

[3] Sailada, Srikanth and Vohra, Neeti and Subramanian, N., "Crystal Dilithium Algorithm For Post Quantum Cryptography: Experimentantion and Usecase for eSign", en *First International Conference on Electrical, Electronics, Information and Communication Technologies*, 2022.

[4] B. Scarone Etchamendi, "Criptografía Post Cuántica basada en Reticulados", Trabajo fin de grado, Universidad de la República de Uruguay, 2018.

[5] Merino González, Luis M., and Santos Aláez, Evangelina. "Algebra Lineal Con Métodos Elementales", Madrid: Thomson Paraninfo, 2006.

[6] D. P. Chi, J. W. Choi, J. S. Kim, and T. Kim, "Lattice Based Cryptography for Beginners," Cryptology ePrint Archive, Paper 2015/938, 2015. [Online]. Disponible en: https://eprint.iacr.org/2015/938.

[7] A. Z. Zahid, "Lattices, Cryptography, and NTRU", Trabajo fin de grado, St. Mary's College of California, 2017.

[8] S. Harrigan, "Lattice-Based Cryptography and the Learning with Errors Problem," , 2017. [En línea]. Disponible en: https://mysite.science.

uottawa.ca/mnevins/papers/StephenHarrigan2017LWE.pdf.

[9] A. Miguel Salgado, "Criptografía Postcuántica", Trabajo fin de grado, Universidad del País Vasco, 2021.

[10] E. Ma, "Online SHAKE-128 Hash Generator", Disponible en: https://emn178.github.io/online-tools/shake_128.html, Consultado en: Abril 14, 2023.

[11] Kaggle, "Kaggle". Disponible en: https://www.kaggle.com/. Consultado en: Marzo 8, 2023.

[12] E. Pérez Ramos, "Implementación CRYSTALS-Dilithium". Disponible en: https://shre.ink/Qw1d. Consultado en: Mayo 21, 2023.

Study and Implementation of the CRYSTALS-Dilithium Signature Scheme

Édgar Pérez Ramos
Facultad de Ciencias • Sección de Matemáticas
Universidad de La Laguna
alu0101207667@ull.edu.es

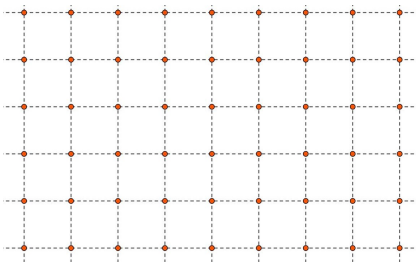
Abstract

CRYSTALS-Dilithium is one of the three digital signature schemes included in the third round of Post-Quantum Cryptography Standardization program of the National Institute of Standards and Technology (NIST). It is a lattice-based algorithm based on the well-known Fiat-Shamir scheme. Specifically, its security is based on the difficulty of the problem of finding short vectors in lattices. This paper provides an introduction to the underlying concepts of the protocol and presents its implementation with a clear educational objective.

1. Introduction

CRYSTALS-Dilithium is based on the theory of lattices. Below, the definition of lattices is introduced. Let V be a vector space over K, where K is a field, and let $B = \{v_1, v_2, ..., v_n\}$ be a basis for a subspace of V. Let A be a ring contained in K. Then the lattice $L \subset V$ generated by $\{v_1, v_2, ..., v_n\}$ is defined as the set:

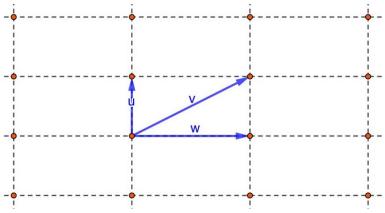
$$\mathcal{L}(v_1, v_2, \ldots, v_n) = \{ \sum_{i=1}^{n} a_i \cdot v_i : a_i \in A \}$$



2. Lattices and Lattice problems

The Shortest Vector Problem and the Learning With Errors Problem are essential in the CRYSTALS-Dilithium scheme.
Given a basis $B \in \mathbb{Z}^{m \times n}$ of a lattice $\mathcal{L}$, the Shortest Vector Problem is defined as follows:

- Search: Finding a vector $v \in \mathcal{L} \setminus \{0\}$ such that $\| v \| = \lambda_1(\mathcal{L})$.
- Optimization: Finding $\lambda_1(\mathcal{L})$.
- Decision: Given a rational number $r \in \mathbb{Q}$, determine if $\lambda_1(\mathcal{L}) \leq r$.



On the other hand, let $n, m, q \in \mathbb{Z}$, the Learning with Errors Problem is based on finding a vector $s \in \mathbb{Z}_q^n$ such that:

$$b = As + e$$

where $A \in \mathbb{Z}_q^{m \times n}$ and $b, e \in \mathbb{Z}_q^n$. Furthermore, The vector e is the error vector and is given by the probability distribution $\mathcal{X}^m$ in $\mathbb{Z}_q^m$.

3. CRYSTALS-Dilithium

CRYSTALS-Dilithium is a digital signature scheme based on public key cryptography. A public key signature scheme Sig is a triad of algorithms executable in polynomial time $Sig = (\mathcal{K}, \mathcal{F}, \mathcal{V})$. Two signature schemes are studied, the *Template* version, which is the initial one, and the final version. For the study of the latter, hash functions, modular reductions, the use of seeds and the Number Theoretical Transformation are studied in depth.

Gen
01 $\mathbf{A} \leftarrow R_q^{k \times \ell}$
02 $(\mathbf{s}_1, \mathbf{s}_2) \leftarrow S_\eta^\ell \times S_\eta^k$
03 $\mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$
04 **return** $(pk = (\mathbf{A}, \mathbf{t}), sk = (\mathbf{A}, \mathbf{t}, \mathbf{s}_1, \mathbf{s}_2))$

Sign(sk, M)
05 $\mathbf{z} := \perp$
06 **while** $\mathbf{z} = \perp$ do
07 $\quad \mathbf{y} \leftarrow S_{\gamma_1 - 1}^\ell$
08 $\quad \mathbf{w}_1 := \text{HighBits}(\mathbf{A}\mathbf{y}, 2\gamma_2)$
09 $\quad c \in B_\tau := \text{H}(M \parallel \mathbf{w}_1)$
10 $\quad \mathbf{z} := \mathbf{y} + c\mathbf{s}_1$
11 $\quad$ **if** $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or $\|\text{LowBits}(\mathbf{A}\mathbf{y} - c\mathbf{s}_2, 2\gamma_2)\|_\infty \geq \gamma_2 - \beta$, then $\mathbf{z} := \perp$
12 **return** $\sigma = (\mathbf{z}, c)$

Verify$(pk, M, \sigma = (\mathbf{z}, c))$
13 $\mathbf{w}_1' := \text{HighBits}(\mathbf{A}\mathbf{z} - c\mathbf{t}, 2\gamma_2)$
14 **if return** $[\![\|\mathbf{z}\|_\infty < \gamma_1 - \beta]\!]$ **and** $[\![c = \text{H}(M \parallel \mathbf{w}_1')]\!]$

To conclude the work, a Python implementation is presented. Both complementary and main functions have been implemented. Also, in order to provide a more didactic work, some changes are made with respect to the *Template* version.
In addition, two experiments with random strings have been performed to empirically verify that there are no authentication problems.

```
#### CRYSTALS-Dilithium ####

q = 2**23- 2**13 + 1
n = np.random.randint(5,15)

pk, sk = keygen(n,n,q)
sigma, beta = sign(sk, "Message")
verify(pk, "Message", sigma, beta)
```

```
True
```

References

[1] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, y D. Stehlé, “CRYSTALS-Dilithium, Algorithm Specifications and Supporting Documentation (Version 3.1)”, 2022. Available on: https://acortar.link/clYOJY