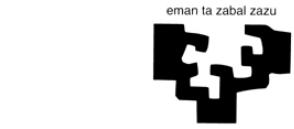

ZTF-FCT
Zientzia eta Teknologia Fakultatea
Facultad de Ciencia y Tecnología

Criptografía Postcuántica

Trabajo Fin de Grado
Grado en Matemáticas

Alba Miguel Salgado

Trabajo dirigido por
Oscar Lage Serrano
María Asunción García Sánchez

Leioa, 17 de Julio de 2021

Índice general

Agradecimientos **v**

Prefacio **vii**

1. Introducción a la Criptografía Postcuántica **1**
1.1. Introducción 1
1.2. Algunos problemas matemáticos no polinomiales y su aplicación a sistemas criptográficos 2
1.2.1. Sistema criptográfico RSA 4
1.2.2. Sistema criptográfico ElGamal 6
1.2.3. Criptosistemas de curva elíptica 8
1.3. Impacto de la computación cuántica en los algoritmos criptográficos 13

2. Esquema de cifrado CRYSTALS-KYBER **17**
2.1. Conceptos matemáticos utilizados 17
2.1.1. Anillos de polinomios 17
2.1.2. Retículos 21
2.2. Problema en el que está basado el esquema de Crystals-Kyber 22
2.2.1. Problema del Vector Más Corto (SVP) 23
2.2.2. Problema de Aprendizaje con Error (LWE) 24
2.3. Descripción del sistema y algoritmos 25

3. Análisis y comparativa **33**
3.1. Descripción del proceso 33
3.2. Resultados de Crystals-Kyber 34
3.2.1. Generación de clave 35
3.2.2. Cifrado del mensaje 35
3.2.3. Descifrado del mensaje 36
3.3. Resultados de los finalistas y comparativa 37
3.3.1. Generación de clave 37
3.3.2. Cifrado del mensaje 38
3.3.3. Descifrado del mensaje 39
3.3.4. Conclusión de la comparativa 39

A. Programación **41**

Bibliografía **53**

Agradecimientos

En primer lugar me gustaría agradecer a mis directores de TFG, María Asunción García y Oscar Lage, el guiarme y ayudarme en todo lo que he necesitado.

Por otro lado, quiero dar las gracias a Tecnalia por brindarme esta oportunidad, así como a Sergio e Iñaki por ayudarme en todo aquello que era nuevo para mí.

También me gustaría dar las gracias a mi familia por su apoyo incondicional, pero sobre todo por mostrar siempre plena confianza en mí. A mi abuelo por enseñarme desde pequeña a respetar y a querer a las matemáticas. A mis amigas, en especial a mis compañeras en este duro viaje, porque sola nunca lo hubiera conseguido.

Por último, gracias a mi profesora de bachiller Hostaizka, que me mostró desde un primer momento con total claridad lo que me iba a encontrar en esta carrera.

Prefacio

El Instituto Nacional de Estándares y Tecnología (NIST) es el encargado de buscar una solución para todos aquellos problemas que se van generando a medida que las nuevas tecnologías van avanzando. En estos momentos, el NIST se encuentra en un proceso de Normalización Postcuántica, que tiene como objetivo la elaboración de normas y directrices para la seguridad de los sistemas. Las dos características principales que se buscan son la seguridad y la eficiencia, ya que si un protocolo no es seguro, no tiene sentido usarlo para proteger los datos, de igual manera que si no es eficiente, perderíamos la rapidez a la que estamos acostumbrados y se ralentizarían las comunicaciones por Internet.

El proceso fue iniciado en 2016 con una convocatoria abierta para impulsar la creación de nuevos algoritmos criptográficos de clave pública resistentes a los ataques de los ordenadores cuánticos. A finales del 2017, el NIST anunció los algoritmos de la primera ronda, en la que se aceptaron un total de 69 candidatos. Tras un cribado selectivo, en enero de 2019 se terminó la primera ronda y se anunciaron los 26 esquemas que habían pasado a la segunda ronda. En estos momentos, finales de 2020, el proceso se encuentra en la tercera ronda que cuenta con 7 finalistas de los cuales 3 son esquemas de firmas digitales y 4 son cifrados de clave pública, siendo estos últimos en los que nos fijamos a continuación:

- **Classic McEliece**
  Se trata de una versión más moderna del esquema de cifrado publicado por McEliece a finales de los años 70, criptosistema de la familia de los basados en códigos. Su seguridad se basa en el problema de la decodificación del síndrome, el cual nos dejaba la desventaja de que a pesar de ser rápido en el cifrado, llevarlo a la práctica no es algo nada sencillo debido al gran tamaño de claves que se requieren.

- **CRYSTALS-KYBER**
  Este esquema pertenece a la familia de los criptosistemas basados en retículos, que utiliza como base un problema llamado *Problema de Aprendizaje con Error (LWE)*. El LWE permite a los cripstosistemas

cuya seguridad puede reducirse a problemas de retículos, llevarlo sobre retículos generales.

- **NTRU**
  Es una variante del criptosistema NTRU que es el primer criptosistema práctico conocido basado en retículos. En este caso, se basa en la dificultad que hay a la hora de resolver los problemas de los retículos dentro de un subgrupo concreto de ellos que incluyen los retículos de la NTRU. Este esquema tiene un gran redimiento en comparación a los de la criptografía clásica, pero tiene un tamaño de clave pública mayor que el de la RSA.

- **SABER**
  El esquema SABER es también un criptosistema basado en red, pero que en este caso su seguridad se basa en la dificultad de resolver el *Problema de aprendizaje con redondeo (LWR)*. Los esquemas basados en LWR se obtiene de forma determinista, lo que hace que se reduzca el tamaño de las claves públicas y de los textos cifrados, además de disminuir el número total de polinomios secretos que deben ser utilizados.

El objetivo de esta memoria será adentrarnos brevemente en el mundo de la Criptografía Postcuántica y estudiar algunos de los avances producidos por el NIST.

Para ello, deberemos comenzar repasando algunos de los más famosos criptosistemas, además de conocer el concepto de criptografía cuántica, así como la relación entre ellos, que es lo que se podrá obervar en el Capítulo 1.

En el Capítulo 2 nos centraremos en estudiar el funcionamiento de uno de los 4 finalistas previamente descritos, concretamente, el esquema de Crystals-Kyber. Para ello, tendremos que recordar ciertos aspectos matemáticos relevantes en él, con el fin de entender y analizar los problemas en los que basa su seguridad. Además, veremos también los algoritmos empleados en la creación de claves, así como los de cifrado y descifrado.

Para finalizar, el Capítulo 3 se enfocará en realizar un análisis práctico del esquema de Crystals-Kyber y compararlo con el resto de finalistas del NIST.

Las referencias básicas que hemos consultado se encuentran recogidas en la Bibliografía que figura al final del documento.

Capítulo 1

Introducción a la Criptografía Postcuántica

El objetivo de este primer capítulo es conocer el concepto de *Criptografía Postcuántica*. Para entenderlo mejor recordaremos algunos fundamentos de la criptografía, así como algunos de los más famosos criptosistemas.

1.1. Introducción

En la actualidad, vivimos en un mundo ampliamente conectado donde la privacidad de las personas en Internet y, sobre todo, la seguridad en las comunicaciones es tan esencial que la criptografía se ha vuelto indispensable en nuestras vidas. Esto es debido a que los principales protocolos de comunicación encargados de proteger los datos se basan en acciones criptográficas básicas, como pueden ser el cifrado de clave pública, las firmas digitales o el intercambio de claves. La seguridad de éstos depende de la dificultad a la hora de resolver ciertos problemas matemáticos que, actualmente, las computadoras clásicas no son capaces de resolver en un tiempo breve. Pero esto puede cambiar con la llegada de las llamadas computadoras cuánticas, máquinas que utilizan la mecánica cuántica, aprovechando las propiedades físicas de la materia y la energía para realizar cálculos a gran velocidad y resolver así dichos problemas.

Un algoritmo desarrollado por Peter Shor en 1994 [7, Capítulo 5], demostró que los ordenadores cuánticos podrían ser capaces de resolver de manera eficiente problemas como la factorización de grandes enteros, pilar fundamental de la seguridad del **criptosistema RSA**, o de romper los **criptosistemas de curva elíptica**, o el **sistema criptográfico de El Gamal**, entre otros muchos.

Por todo lo anterior, es evidente que los ordenadores cuánticos suponen

una gran amenaza a todas estas técnicas para ocultar datos, por lo que es indispinsable pensar en una solución. Para ello contamos con la *criptografía postcuántica* o *criptografía cuántica resistente*, que se encargará de desarrollar sistemas criptográficos seguros ante las computadoras cuánticas, que además sean compatibles con los protocolos y las redes de comunicación ya existentes.

1.2. Algunos problemas matemáticos no polinomiales y su aplicación a sistemas criptográficos

Supongamos que un emisor A desea enviarle un mensaje m a un receptor B por un canal que no es seguro. Para que solo B sea quien lo lea, A encripta su mensaje mediante una clave k, produciendo un texto cifrado que B recibe y que descifra utilizando el proceso contrario al encriptado. Si A y B acuerdan una clave común, a usar en los procesos de cifrado y descifrado, esto es lo que denominamos como *criptografía de clave privada o simétrica*.

Este tipo de criptografía nos permite tener un método seguro por el que enviar el mensaje a través de un canal no seguro. Sin embargo, en el caso de la criptografía simétrica debemos tener en cuenta que se utiliza la misma clave tanto para cifrar como para descifrar el mensaje, lo cual puede llegar a generar una serie de problemas. Por ejemplo, tenemos los siguientes:

(i) **Distribución de la clave**
El emisor y el receptor deben acordar la clave a usar para lo cual necesitan un sistema seguro, ya que no pueden hacerlo por el canal directamente.

(ii) **Número de claves**
Si el colectivo de personas que se quieren comunicar es muy grande, cada par de usuarios necesitará un par de claves separadas en una red con n usuarios; esto nos deja un total de $\frac{n(n-1)}{2}$ par de claves, teniendo cada usuario que almacenar $n-1$ claves distintas para comunicarse con los $n-1$ miembros restantes de la red.

Para tratar de solucionar estas dificultades surge la *criptografía de clave pública o asimétrica*. Es necesario entender que el verdadero peligro se encuentra en la parte de descifrar, ya que el hecho de que se conozca la manera de encriptar un mensaje no supone ninguna amenza. Por ello, cabe la posibilidad de que cada usuario disponga de una clave que conste de dos partes: una parte que hace publica, k_{pub} y que se utiliza en el proceso de

cifrado de los mensajes que va a recibir, y otra parte, que mantiene privada, k_{priv}, que usa en el proceso de descifrado. Así pues, cuando un individuo A quiere mandar un mensaje a B, consulta la clave pública de B y la usa para encriptar su mensaje. Una vez que B recibe el mensaje cifrado, utiliza su clave privada para descifrarlo.

La seguridad de esta se basa principalmente en la dificultad a la hora de resolver ciertos problemas matemáticos, además de en la complejidad de adivinar la clave utilizada. Teniendo ésto en cuenta podemos clasificar algunos algoritmos de clave pública en los siguientes grupos:

(i) **Esquemas de factorización de números enteros**
Son aquellos que se basan en la dificultad de factorizar números enteros de gran tamaño. El esquema que más destaca dentro de esta familia es el *criptosistema RSA* [19], aunque podemos encontrar otros como puede ser el *criptosistema Rabin* [13].

(ii) **Esquemas de logaritmo discreto**
Son los basados en el problema del logaritmo discreto, el cual nos habla de la dificultad de localizar el valor de $m \in \mathbb{Z}$ trabajando en $\mathbb{Z}_n$, donde conocemos el valor de x para $x \equiv a^m \bmod n$ con $a \in \mathbb{Z}$. Uno de los ejemplos más importantes es *El Gamal* [5].

(iii) **Esquemas de curva elíptica (EC)**
Se trata de una generalización del problema del logaritmo discreto, en el cual se emplean curvas elípticas, ya que para los conjuntos de puntos en estas curvas usados en estos criptosistemas es aún más difícil de resolver que en el caso de los cuerpos finitos del problema original, lo que le aporta mayor seguridad. En esta familia nos podemos encontrar con los *criptosistemas de curva elíptica* [23].

Definición 1.2.1. Sea $\mathcal{M}$ un conjunto de mensajes claros, $\mathcal{K}$ un conjunto de claves y $\mathcal{C}$ un conjunto de mensajes cifrados. Se llama **función de encriptado o cifrado** a una aplicación $e : (\mathcal{K}, \mathcal{M}) \longrightarrow \mathcal{C}$ y **función de desencriptado o descifrado** a una aplicación $d : (\mathcal{K}, \mathcal{C}) \longrightarrow \mathcal{M}$. Además, $(\mathcal{M}, \mathcal{K}, \mathcal{C}, e, d)$ constituye un **esquema o sistema criptográfico** si las funciones d y e verifican

$$d(k, e(k, m)) = m \quad \forall (k, m) \in \mathcal{K} \times \mathcal{M}$$

Proposición 1.2.1. *Sea $(\mathcal{M}, \mathcal{K}, \mathcal{C}, e, d)$ un sistema criptográfico y $k_1 \in \mathcal{K}$. Entonces, tenemos las funciones*

$$e_{k_1} : \begin{array}{ccc} \mathcal{M} & \longrightarrow & \mathcal{C} \\ m & \longmapsto & e(k_1, m) \end{array} \qquad d_{k_1} : \begin{array}{ccc} \mathcal{C} & \longrightarrow & \mathcal{M} \\ c & \longmapsto & d(k_1, m) \end{array}$$

verfican

$$d_{k_1}(e_{k_1}(m)) = m \quad \forall m \in \mathcal{M} \tag{1.1}$$

donde e_{k_1} *es una aplicación inyectica para todo* $k_1 \in \mathcal{K}$. *Además, si restringimos* d_{k_1} *a* $e_{k_1}(\mathcal{M})$ *entonces* $d_{k_1}|_{e_{k_1}(\mathcal{M})}$ *es la aplicación inversa de* e_{k_1}.

Observación 1.2.1. Recordemos que en el caso de la criptografía de clave pública, cada clave consta de dos partes; una privada y otra pública, usándose la parte privada en el proceso de desencriptado y la pública en el de encriptado, quedando así las funciones $e_{k_{pub}}$ y $d_{k_{priv}}$.

A continuación, estudiaremos brevemente alguno de los ejemplos descritos en cada tipo de esquema de clave pública mencionados anteriormente.

1.2.1. Sistema criptográfico RSA

El esquema criptográfico RSA [19], llamado así por las iniciales de los apellidos de sus creadores (Ronald Rivest, Adi Shamir y Leonard Adleman), se basa en la dificultad de factorizar un número natural grande como producto de primos.

Tanto la encriptación, como la desencriptación de RSA, se trabaja desde el anillo de enteros $\mathbb{Z}_n = \{\bar{a} \mid \bar{a} \in \{0, ..., n-1\}$ siendo $\bar{a} = \{a + nz \mid z \in \mathbb{Z}\}\}$. Veamos cuales son las etapas de este sistema:

- **Generación de claves**
  Como ya hemos comentado antes, las claves k de un sistema critográfico asimétrico son pares $k = (k_{pub}, k_{priv})$. Para contruir la clave, el individuo B elige dos primos grandes p, q y los multiplica para obtener $n = pq$; además, escoge al azar un número $e \in \mathbb{N}$ tal que $(e, \varphi(n)) = 1$ $e < \varphi(n)$, siendo $\varphi(n) = \varphi(pq) = (p-1)(q-1)$ la **Función de Euler**, que recordemos que nos devuelve el número de naturales menores que n, que sean coprimos con n. Por último, calcula el valor de d tal que $d \cdot e = 1 \bmod \varphi(n)$, en lo que podemos observar la importancia de que $(e, \varphi(n)) = 1$ que asegura que existe $e^{-1} = d \in \mathbb{Z}_{\varphi(n)}$. Teniendo todo esto en cuenta B construye $k_{pub} = (n, e)$ clave que publica y $k_{priv} = (p, q, d)$ clave que mantiene en secreto.

- **Cifrado del mensaje**
  Cuando el individuo A quiere mandarle un mensaje m a B, consulta $k_{pub} = (n, e)$ de B y cifra el mensaje, para posteriormente enviarselo, mediante la función de encriptado que en este caso es de la siguiente forma:

$$\begin{array}{rccc} e_{k_{pub}}: & \mathbb{Z}_n & \longrightarrow & \mathbb{Z}_n \\ & m & \longmapsto & m^e \bmod n \end{array}$$

- **Descifrado del mensaje**
  Una vez que el usuario B recibe el mensaje cifrado c, deberá descifrarlo utilizando su clave privada $k_{priv} = (p, q, d)$, recordando que $n = pq$, mediante la función de desencriptado

$$\begin{aligned} d_{k_{priv}} : \ \mathbb{Z}_n &\longrightarrow \mathbb{Z}_n \\ c &\longmapsto c^d \bmod n \end{aligned}$$

Ahora bien, veamos si el sistema criptográfico RSA está bien definido. Para ello, sabemos que tiene que verificar (1.1) con las funciones que acabamos de definir. Así pues, desarrollando la primera parte, quedaría

$$d_{k_{priv}}(e_{k_{pub}}(m)) = d_{k_{priv}}(m^e) = (m^e)^d = m^{ed} \bmod n$$

por lo que faltaría ver que se cumple

$$m^{ed} = m \bmod n$$

y así obtener que $d_{k_{priv}}(e_{k_{pub}}(m)) = m$.

Para ello, empezamos por recordar que $ed = 1 \bmod \varphi(n)$, o lo que es lo mismo, $ed = 1 + t \cdot \varphi(n)$ para $t \in \mathbb{Z}$. Es necesario distiguir dos casos:

1. Si $(m, n) = 1$

   $$d_{k_{priv}}(e_{k_{pub}}(m)) = m^{ed} = m^{1+t\cdot\varphi(n)} = m \cdot m^{t\cdot\varphi(n)} \equiv (m^{\varphi(n)})^t m \bmod n$$

   y por el teorema de Euler [17, Sección 6.3.4], $m^{\varphi(n)} \equiv 1 \bmod n$, luego

   $$(m^{\varphi(n)})^t m \equiv 1^t m \equiv m \bmod n$$

2. Si $(m, n) \neq 1$

   2.1. Con $m \equiv 0 \bmod n$

   $$m \equiv 0 \bmod n \Longrightarrow m^{ed} \equiv 0 \bmod n \Longrightarrow m^{ed} \equiv m \bmod n$$

   2.2. Con $m \not\equiv 0 \bmod n$
   En este caso, o $(m, n) = p$ o $(m, n) = q$, por ser $n = pq$ con p, q dos primos distintos. Supongamos sin pérdida de generalidad que $(m, n) = p$, entonces $(m, p) = p$ y $(m, q) = 1$, por ser $(p, q) = 1$. Por tanto, podemos utilizar nuevamente el teorema de Euler para $m^{\varphi(q)} \equiv 1 \bmod q$, y como $\varphi(n) = \varphi(q)\varphi(p)$, entonces:

   $$(m^{\varphi(q)})^{\varphi(p)} \equiv 1 \bmod q \Longrightarrow m^{\varphi(n)} \equiv 1 \bmod q \Longrightarrow$$

   $$\Longrightarrow (m^{\varphi(n)})^t m \equiv m \bmod q \Longrightarrow m^{ed} \equiv m \bmod q$$

Por otro lado, como $(m, n) = p$ tenemos que

$$m \equiv 0 \bmod p \Longrightarrow m^{ed} \equiv 0 \bmod p \Longrightarrow m^{ed} \equiv m \bmod p$$

y teniendo en cuenta ambas expresiones, así como el Teorema chino de los restos [20, Sección 1.3.4], conseguimos:

$$m^{ed} \equiv m \bmod pq \Longleftrightarrow m^{ed} \equiv m \bmod n$$

Ejemplo 1.2.1. Consideramos los primos $p = 53$ y $q = 79$, además de $e = 137$, por lo que $k_{pub} = (53, 79, 137)$. Además tenemos que $n = pq = 53 \cdot 79 = 4187$. Para ver cuál sería su clave privada, basta con calcular el valor de d tal que $d \cdot e = 1 \bmod \varphi(n)$ siendo $\varphi(4187) = (53-1)(79-1) = 4056$. Así pues, calculamos el inverso de 137 en $\mathbb{Z}/4056\mathbb{Z}$, que es $d = 977$, quedando así $k_{priv} = (4187, 977)$.
Ahora bien, si un emisor A quisiera enviar un mensaje m a un receptor B, como por ejemplo, $m = 826\ 437\ 109$, tomaría $k_{pub} = (53, 79, 137)$ y aplicaría la función de cifrado para cada cifra del mensaje:

$$\begin{cases} 826^{137} \bmod 4187 \Longrightarrow 73 \\ 437^{137} \bmod 4187 \Longrightarrow 3566 \\ 109^{137} \bmod 4187 \Longrightarrow 3325 \end{cases}$$

Por lo que nos queda el mensaje encriptado $c = 73\ 3566\ 3325$.
Por otro lado, veamos como descifraría el individuo B ese mensaje encriptado obteniendo el mensaje claro m. Teniendo en cuenta su $k_{priv} = (4187, 977)$ y aplicando la función de desencriptado, obtenemos

$$\begin{cases} 73^{977} \bmod 4187 \Longrightarrow 826 \\ 3566^{977} \bmod 4187 \Longrightarrow 437 \\ 3325^{977} \bmod 4187 \Longrightarrow 109 \end{cases}$$

dejando así el mensaje claro $m = 826\ 437\ 109$.

1.2.2. Sistema criptográfico ElGamal

El esquema criptográfico de ElGamal [5] fue descrito por el criptógrafo egipcio Taher Elgamal en 1985. Como hemos comentado anteriormente, basa su seguridad en la complejidad sobre el problema matemático del logaritmo discreto.

Para este caso se trabaja sobre $\mathbb{Z}_p$, donde p es un número primo. Las estapas de este sistema son las siguientes:

- **Generación de claves**
  Por un lado, el individuo A toma un generador g del grupo cíclico $(\mathbb{Z}_p)$

y un número aleatorio $a \in \{0, ...p-1\}$. Además, calcula $t \equiv g^a \bmod p$, lo que nos deja una clave pública $k_{pub} = (p, g, t)$ y una clave privada $k_{priv} = a$.

- **Cifrado del mensaje**
  Supongamos que un individuo B quiere mandar un mensaje m a un individuo A. Para ello, consulta la clave pública de A y elige al azar un número $b \in \{2, ..., p-2\}$ y calcula

$$\begin{cases} c_1 \equiv g^b \bmod p \\ c_2 \equiv t^b m \bmod p \end{cases}$$

  lo que hace que obtenga el mensaje cifrado $c = (c_1, c_2)$ el cual envía al individuo A.

- **Descifrado del mensaje**
  Cuando el individuo A recibe el mensaje cifrado c, comienza calculando el inverso de c_1^a modulo p utilizando su clave privada. Llamamos a ese valor $u = c_1^a \bmod p$. Así pues obtiene la decodificación del mensaje como

$$m = c_2 u^{-1} \bmod p$$

En este caso, es bastante sencillo ver este criptosistema está bien definido, es decir, que verifica (1.1):

$$\begin{aligned} d_{k_{priv}}(e_{k_{pub}}(m)) &= d_{k_{priv}}((c_1, c_2)) = c_2 u^{-1} \equiv c_2 (c_1^a)^{-1} \equiv t^b m[(g^b)^a]^{-1} \\ &\equiv m(g^a)^b[(g^b)^a]^{-1} \equiv m g^{ab}(g^{ab})^{-1} \equiv m \bmod p \end{aligned}$$

Ejemplo 1.2.2. Los individuos A y B deciden utilizar el sistema criptográfico de ElGamal para enviar mensajes de forma segura. Para ello A escoge un número primo grande $p = 7286131$ y como generador $g = 5$, además de un número aleatorio $a = 13579$ para tener así que $k_{priv} = 13579$. Para completar la clave pública que va a utilizar, calcula $t \equiv g^a \bmod p \Rightarrow t \equiv 5^{13579} \bmod 7286131$, por lo que $t = 3511742$ y, por tanto, publica $k_{pub} = (7286131, 5, 3511742)$.
Supongamos ahora que el individuo B quiere enviarle el mensaje $m = 31730$ a A. Para ello, toma $b = 2468$, consulta $k_{pub} = (7286131, 5, 3511742)$ y calcula

$$c_1 = g^b \bmod p \Longrightarrow c_1 = 5^{2468} \bmod 7286131 \Longrightarrow c_1 = 4019697$$

$$c_2 = t^b m \bmod p \Rightarrow c_2 = 3511742^{2468} \cdot 31730 \bmod 7286131 \Rightarrow c_2 = 2524279$$

De esta manera, envía el mensaje cifrado $c = (4019697, 2524279)$.
Por otro lado, cuando el individuo A reciba el mensaje cifrado c, utilizará su clave privada para desencriptarlo de la siguiente manera:

$$u = c_1^a \bmod p \Longrightarrow u = 4019697^{13579} \bmod 7286131 \Longrightarrow u = 298827$$

Ahora bien, el inverso de u modulo 7286131 es 200765, por lo que se obtiene que

$$m = c_2 u^{-1} \bmod p \Longrightarrow m = 2524279 \cdot 200765 \bmod 7286131 \Longrightarrow m = 31730$$

1.2.3. Criptosistemas de curva elíptica

La criptografía de curvas elípticas introducida por Victor Miller [14] y Neal Koblitz [9], se basa en la resolución de un análogo al problema del logaritmo discreto, para el cual utiliza las curvas elípticas en un cuerpo finito como base de un grupo.

Definición 1.2.2. Una **curva elíptica** es un conjunto de puntos dados en un cuerpo F que satisface la siguiente ecuación:

$$y^2 + a_1xy + a_2y = x^3 + a_3x^2 + a_4x + a_5 \tag{1.2}$$

donde $a_1, ..., a_5 \in F$.

La definición anterior se puede simplificar si trabajamos en un cuerpo que no sea de característica 2 ó 3. Recordemos que la característica de un cuerpo F, $char(F)$, es el menor número n que cumpla que sumando 1 n veces obtengamos cero.

Proposición 1.2.2. *Sea F un cuerpo con $char(F) \neq 2, 3$. Entonces podemos simplificar la ecuación (1.2) a*

$$y^2 = x^3 + ax + b$$

donde $a, b \in F$. Este tipo de ecuación es conocida como la ***forma normal de Weierstrass****.*

Demostración. Supongamos que tenemos una curva elíptica de ecuación (1.2). Teniendo en cuenta que $char(F) \neq 2$, podemos dividir entre dos y agruparlo de modo que:

$$\left(y + \frac{a_1x}{2} + \frac{a_2x}{2}\right)^2 = x^3 + \left(a_3 + \frac{a_1^2}{4}\right)x^2 + \left(a_4 + \frac{a_1a_2}{2}\right)x + \left(a_5 + \frac{a_2^2}{4}\right)$$

Haciendo ahora los siguiente cambios de variables

$$y_0 = y + \frac{a_1x}{2} + \frac{a_2x}{2}$$

$$a_3' = a_3 + \frac{a_1^2}{4}$$

$$a_4' = a_4 + \frac{a_1a_2}{2}$$

$$a'_5 = a_5 + \frac{a_2^2}{4}$$

obtenemos

$$y_0^2 = x^3 + a'_3x^2 + a'_4x + a'_5$$

Finalmente, y operando de manera similar teniendo en cuenta que $char(F) \neq 3$, podemos completar cubos tomando $x_0 = x + \frac{a'_3}{3}$ obteniendo así la expresión que queríamos:

$$y_0^2 = x_0^3 + ax_0 + b$$

□

Observación 1.2.2. Un ejemplo de cuerpo con $char(F) \neq 2, 3$ es $\mathbb{R}$, y un ejemplo de curva elíptica sobre $\mathbb{R}$ es $y^2 = x^3 - 5x + 7$ que podemos observar dibujada en Figura 1.1.

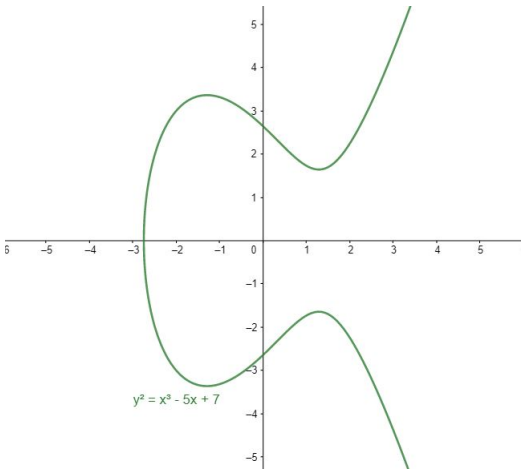

Figura 1.1: Gráfica de la curva elíptica $y^2 = x^3 - 5x + 7$ en $\mathbb{R}^2$

Sabemos que una curva elíptica $y^2 = x^3 + ax + b$ en $\mathbb{R}^2$ es simétrica con respecto al eje de abscisas debido a que para cualquier x_i, que pertenezca a la curva, tanto $y_i = \sqrt{x_i^3 + ax_i + b}$ como $y'_i = -\sqrt{x_i^3 + ax_i + b}$ son soluciones de la ecuación. Por tanto, si tomamos un punto $P = (x_1, y_1)$ que pertenezca a la curva, el punto $P' = (x_1, -y_1)$ también estará en ella.

Algebraicamente, la suma de puntos de una curva elíptica se define de la siguiente forma:

Sean $P = (x_1, y_1)$ y $Q = (x_2, y_2)$ dos puntos de la curva elíptica. Definimos un tercer punto $R = (x_3, y_3)$ el cual será el punto de intersección entre

la recta que pasa por los puntos P y Q y la curva elíptica:

$$\begin{cases} y = \dfrac{(y_2 - y_1)x + y_1x_2 - y_2x_1}{x_2 - x_1} \\ y^2 = x^3 + ax + b \end{cases}$$

Este punto R se denomina *suma* de P y Q, y se denota como $P + Q = R$. Puede encontrarse en tres situaciones diferentes:

- $P \neq Q$ y $P \neq P'$
  La coordenadas de $R = (x_3, y_3)$ se expresan como

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_2 - x_1$$
$$y_3 = \frac{y_2 - y_1}{x_2 - x_1}(x_1 - x_3) - y_1$$

  En este caso, el punto R se obtiene al reflejar, con respecto al eje X, el punto obtenido en la intersección entre la recta que pasa por los puntos P y Q y la curva elíptica.

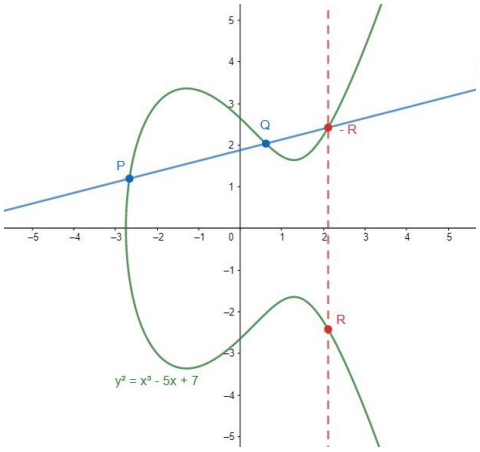

Figura 1.2: Curva elíptica $y^2 = x^3 - 5x + 7$ en $\mathbb{R}^2$ tomando P, Q puntos diferentes

- $Q = P$
  En este caso, $R = P + P = 2P$, y por tanto, las coordenadas de $R = (x_3, y_3)$ se expresan como

$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1$$
$$y_3 = \frac{3x_1^2 + a}{2y_1}(x_1 - x_3) - y_1$$

  Para esta situación necesitamos una construcción geométrica algo diferente ya que la recta que debemos dibujar es la tangente al punto P.

Nuevamente obtendremos el punto R, tal y como está definido, como la reflexión del punto obtenido en la intersección con respecto al eje X entre esa recta y la curva.

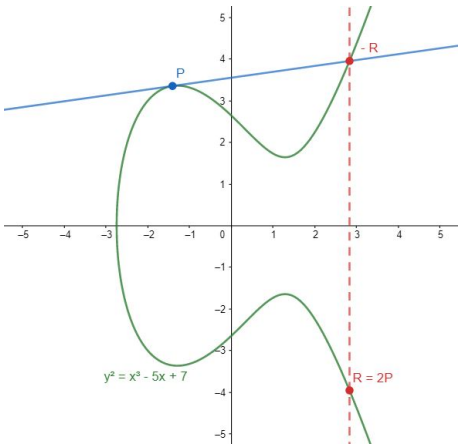

Figura 1.3: Curva elíptica $y^2 = x^3 - 5x + 7$ en $\mathbb{R}^2$ tomando el mismo punto $Q = P$

- $Q = -P$
  En este caso, $P + (-P) = \mathcal{O}$, donde $\mathcal{O}$ es el llamado *punto en el infinito*, que es un punto imaginario situado por encima del eje de abscisas a una distancia infinita. Es por ello que en nuestro caso no existe ningun valor concreto como intersección entre la curva y la recta vertical formada por los puntos P y $-P$.

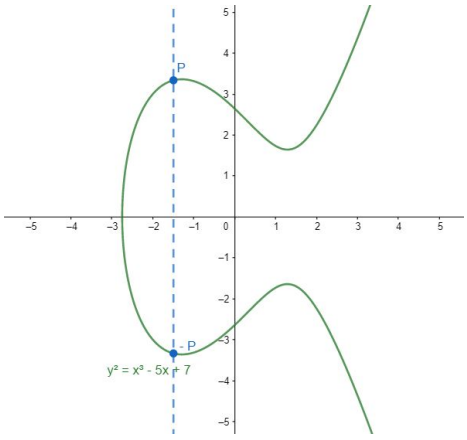

Figura 1.4: Curva elíptica $y^2 = x^3 - 5x + 7$ en $\mathbb{R}^2$ tomando un punto y su opuesto

Observación 1.2.3. Dado el conjunto $\mathbb{F}_p$, cuerpo finito con p elementos, siendo un primo p, la definición dada anteriormente en $\mathbb{R}$ para la suma de dos puntos de una curva elíptica, se puede utilizar exactamente de la misma manera. Para este caso la ecuación de la curva elíptica, escrita en forma normal de Weierstrass, quedaría

$$y^2 = x^3 + ax + b \mod p \quad , p \neq 2, 3$$

Teniendo en cuenta todo esto, tenemos que ver su utilidad en la criptografía. Aunque no vamos a demostrarlo, es necesario saber que todos los puntos de una curva elíptica forman un grupo cíclico junto con la suma + tal y como la hemos definido. Esto nos permite construir sistemas criptográficos a partir de dichos grupos cíclicos.

Definición 1.2.3. Sea una curva elíptica y P y T dos puntos de ella. Entonces, llamamos **Problema del Logaritmo Discreto en Curvas Elípticas** a calcular el valor de n tal que

$$\underbrace{P + P + ... + P}_{n} = nP = T$$

En los criptosistemas de curva elíptica, se toma como clave privada, $k_{priv} = n$, y como clave pública $k_{pub} = (P, T)$. Esto es debido a que mientras el punto $T = nP$ es sencillo de calcular, por ejemplo utilizando el **Programa A.3**, el cálculo del valor n sabiendo los puntos P y T, es decir, el Problema del Logaritmo Discreto en Curvas Elípticas al trabajar en $\mathbb{F}_p$, no es nada sencillo de resolver, lo que le da seguridad al criptosistema.

Ahora bien, si quisiéramos encriptar y desencriptar un mensaje, podemos útilizar una combinación de este esquema con, por ejemplo, el de ElGamal, de la siguiente manera:

- **Cifrado del mensaje**
  Supongamos que un individuo B quiere enviar un mensaje a un individuo A. Para ello expresa dicho mensaje como un punto M de la curva elíptica $y^2 = x^3 + ax + b$, previamente acordada por ambos individuos, consulta la clave pública de A y elige al azar un número k para calcular

  $$\begin{cases} c_1 = kP \\ c_2 = M + kT \end{cases}$$

  y obtener el mensaje cifrado $c = (c_1, c_2)$.

- **Descifrado del mensaje**
  Cuando el individuo A recibe el mensaje cifrado c, calcula el mensaje original utilizando su clave privada n de la siguiente manera:

  $$M = c_2 - nc_1$$

Ejemplo 1.2.3. Tomamos la curva elíptica del ejemplo anterior, $y^2 = x^3 - 5x+7$, pero esta vez en el cuerpo $\mathbb{F}_{17}$. Dado un punto de la curva, $P = (15, 3)$ calculamos

$$5P = \underbrace{P + P + ... + P}_{5} = (11, 5) = T$$

ya que

$$\begin{array}{rcccccc} 2P & = & P+P & = & (15,3)+(15,3) & = & (3,11) \\ 3P & = & 2P+P & = & (3,11)+(15,3) & = & (7,3) \\ 4P & = & 3P+P & = & (7,3)+(15,3) & = & (12,14) \\ 5P & = & 4P+P & = & (12,14)+(15,3) & = & (11,5) \end{array}$$

Por tanto, tenemos que $k_{pub} = [(15, 3), (11, 5)]$ y $k_{priv} = 5$.

Imaginemos que un individuo A quiere enviar un mensaje a B y lo representa como el punto $M = (3, 6)$ de la curva $y^2 = x^3 - 5x + 7$ en el cuerpo $\mathbb{F}_{17}$, que recordemos han escogido ambos para realizar sobre ella todas sus transmisiones. Ahora bien, para encriptar M, cogería los puntos publicados por B, $P = (15, 3)$ y $T = (11, 5)$ y elegiría al azar un número $k = 7$ para calcular

$$c_1 = kP \Longrightarrow c_1 = 7P \Longrightarrow c_1 = 7(15, 3) \Longrightarrow c_1 = (11, 12)$$

$$c_2 = M + kT \Longrightarrow c_2 = (3, 6) + 7(11, 5) \Longrightarrow c_2 = (3, 6) + (15, 14) = (7, 14)$$

De esta forma, envía el mensaje cifrado $c = [(11, 12), (7, 14)]$.
Cuando el individuo B reciba el mensaje cifrado c utilizará su clave privada para desencriptarlo, calculando $k_{priv}c_1 = 5(11, 12) = (15, 14)$ y por último, restandole al punto obtenido c_2:

$$M = (7, 14) - (15, 14) = (7, 14) + (15, -14) = (3, 6)$$

1.3. Impacto de la computación cuántica en los algoritmos criptográficos

Desde que se descubrió el algoritmo de Shor, que comentábamos al inicio del capítulo, los algoritmos cuánticos han evolucionado de forma exponencial, generando una gran expectación ante los peligros que puedan ocasionar en los actuales sistemas criptográficos. Especialmente, preocupan los criptosistemas de clave pública, ya que hasta el momento los simétricos aparentan ser resistentes a ataques válidos en una era cuántica. El único ataque conocido se basa en el *algoritmo de búsqueda de Grover*, que ofrece una aceleración cuadrática para los algoritmos de búsqueda cuántica en comparación con los algoritmos de búsqueda de los ordenadores clásicos. Esto a priori no supone una gran amenaza, ya que es fácilmente compensable duplicando el tamaño

de la clave.
Es por ello que los estudios se están centrando en la busqueda de algoritmos asimétricos resistentes a los ataques de las computadoras cuánticas, que podemos dividir en cuatro categorías principales:

- Criptografía basada en retículos
  Los algoritmos basados en retículos son muy rápidos y se consideran de gran seguridad cuántica. El problema que les atañe es el *Problema del Vector Más Corto*, es decir, que al recibir la base de una red, se pide que se encuentre el punto de ella más cercano al origen, para así obtener el vector no nulo más corto dentro de la red. Este tipo de problemas se benefician de que todas las claves son tan difíciles de romper tanto en el caso más sencillo, como en el peor de los casos, por lo que en la criptografía basada en retículos, todas las posibles selecciones de claves son fuertes y complicadas de resolver. A día de hoy, no se conoce ningún algoritmo cuántico capaz de resolver el Problema del Vector Más Corto con la ayuda de un ordenador cuántico. Uno de los ejemplos con mayor interes en este campo es el *sistema Hoffstein-Pipher-Silverman "NTRU" (1998)* [8].

- Criptografía basada en códigos
  Estos algoritmos se basan en la teoría de codificación, que estudia cómo enviar información a traves de un canal poco seguro sin ningún percance. Para ello, utilizan los códigos de corrección de errores, como pueden ser los *códigos Goppa*, que se basan en codificar un mensaje de manera que en él se encuentren una cantidad de datos erróneos que solo el receptor podrá decodificarlo utilizando su código de corrección de errores. Esta técnica es difícil de revertir, utilizando un ordenador clásico o cúantico, ya que es lo que se conoce como el problema de decodificación por síndrome. La gran desventaja de estos algoritmos es que a pesar de ser bastante rápidos, tienen tamaños de claves muy grandes. En este caso, el ejemplo más claro es el sistema de cifrado de clave pública de *McEliece (1978)* [12].

- Criptografía polinómica multivariante
  La criptografía multivariante se sustenta en el problema de resolver sistemas de ecuaciones polinómicas multivariantes. Actualmente, el esquema más prometedor es el *esquema de cifrado de Matriz Simple* en el que todos los cálculos se realizan sobre un cuerpo finito, y a la hora de desencriptar, sólo es necesario encontrar la solución de sistemas lineales. Aunque a lo largo de la historia ha habido varias propuestas de esquemas de cifrado multivariante, es cierto que tiene mayor éxito en firmas. Un ejemplo interesante es el *"HFEv-" de Patarin (1996)*, que generaliza una propuesta de Matsumoto e Imai [18].

- Criptografía basada en isogenias
  En este caso, el problema del logaritmo discreto en las curvas elípticas, en grupos abelianos, puede ser resuelto de manera eficiente por el algoritmo de Shor en un ordenador cuántico. Para crear esquemas resistentes a los ataques cuánticos, se puede mirar a los grupos no abelianos, que es el llamado *problema de la isogenia en las curvas supersingulares*. Una curva supersingular se define sobre el cuerpo $\mathbb{F}_{p^2}$ para algún p primo, y es aquella que no tiene puntos de orden p y que tiene $p \pm 1$ puntos. Además, es necesario conocer que una **isogenia** es una aplicación desde una curva elíptica E hasta otra curva elíptica E', teniendo ambas curvas el mismo número de puntos. Este tipo de esquema está a falta de ser analizado en profundidad para ver si su seguridad es realmente eficiente.

Capítulo 2

Esquema de cifrado CRYSTALS-KYBER

Como comentábamos en el Prefacio, existen actualmente 4 finalistas de cifrados de clave pública. En este capítulo nos centraremos en analizar uno de ellos, concretamente el sistema de cifrado de *CRYSTALS-KYBER*, comenzando por estudiar los aspectos matemáticos relevantes en él, continuando por mostrar los problemas matemáticos en los que se basan y finalizando con una explicación de los algoritmos de la creación de claves, así como los de cifrado y descifrado.

2.1. Conceptos matemáticos utilizados

2.1.1. Anillos de polinomios

Comenzamos recordando algunas definiciones que iremos utilizando a medida que avance el capítulo.

Definición 2.1.1. Sea R un conjunto no vacío con dos leyes de composición interna, que denotaremos $\cdot$ y $+$. Entonces, $(R, +, \cdot)$ es un *anillo* si

i) $(R, +)$ es un grupo abeliano

ii) $(R, \cdot)$ es una ley de composición interna y asociativa

iii) Cumple la propiedad distributiva, es decir, que para todo $a, b, c \in R$ se verfican las siguientes igualdades:

$$\begin{cases} a \cdot (b + c) &= a \cdot b + a \cdot c \\ (b + c) \cdot a &= b \cdot a + c \cdot a \end{cases}$$

Si además, $(R, \cdot)$ cumple la propiedad conmutativa, entonces se dice que R es un *anillo conmutativo*.

Proposición 2.1.1. *Sea R un anillo conmutativo. Entonces el conjunto $R[x]$ de polinomios sobre R, dado como*

$$R[x] = \left\{ \sum_{i=0}^{n} a_i x^i \;\middle|\; n \in \mathbb{N} \cup \{0\},\ a_i \in R,\ \forall i \in \{1, \ldots, n\} \right\}$$

junto con la suma $+$ y la multiplicación $\cdot$ de forma que dados $f(x) = \sum_{i=0}^{n} a_i x^i$, $g(x) = \sum_{j=0}^{m} b_j x^j \in R[x]$ cumple que

(i) *si $n \geq m$ (de forma análoga para $m \geq n$) con $b_j = 0$ para $j \in \{m+1, ..., n\}$, entonces*

$$f(x) + g(x) = \sum_{i=0}^{n} a_i x^i + \sum_{j=0}^{m} b_j x^j = \sum_{k=0}^{n} (a_k + b_k) x^k$$

(ii) $f(x)g(x) = \left(\sum_{i=0}^{n} a_i x^i \right) \left(\sum_{j=0}^{m} b_j x^j \right) = \sum_{k=0}^{n+m} c_k x^k$, *donde* $c_k = \sum_{i+j=k} a_i b_j$

Estas operaciones definen un anillo conmutativo $(R[x], +, \cdot)$ llamado anillo de polinomios en x con coeficientes en R.

Definición 2.1.2. Para cada número natural n, se llama *n-ésimo polinomio ciclotómico* $\Phi_n(x)$, al polinomio

$$\Phi_n(x) = \prod_{\substack{1 \leq k \leq n \\ (k,n)=1}} (x - \omega^k) \ \in \mathbb{Z}[x]$$

donde $\omega^k = e^{\imath \frac{2\pi k}{n}} \in \mathbb{C}$ son las *n-raíces primitivas de la unidad.*

Es fácil ver que

$$x^n - 1 = \prod_{1 \leq k \leq n} (x - \omega^k) = \prod_{d|n} \Phi_d(x)$$

de donde se obtiene la siguiente fórmula de recurrencia

$$\Phi_n(x) = \frac{x^n - 1}{\prod_{\substack{d|n \\ d<n}} \Phi_d(x)}$$

Estos polinomios se usan en el sistema criptográfico de Crystals-Kyber, trabajando en el anillo $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$, con q primo y $n = 2^{n'-1}$.

Por otro lado, es necesario aclarar que los vectores que se utilicen serán vectores columna que además, denotaremos con letras minúsculas,

$$\mathbf{v} = \begin{bmatrix} v_1 \\ \vdots \\ v_i \\ \vdots \\ v_n \end{bmatrix}$$

con el elemento i-ésimo $v_i \in \mathcal{R}_q \; \forall i \in \{1, \cdots, n\}$, así como las letras mayúsculas denotan matrices,

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} \underset{not.}{\equiv} [a_{ij}]$$

con el elemento de la i-ésima fila y j-ésima columna, $a_{ij} \in \mathcal{R}_q \; \forall i \in \{1, \cdots, n\}$, $j \in \{1, \cdots, m\}$.

Para un vector $\mathbf{v} \in \mathcal{R}_q^n$, definimos el vector *transpuesto* de $\mathbf{v}$ que denotamos como $\mathbf{v}^T$, al vector fila de $\mathbf{v}$, es decir,

$$\mathbf{v}^T = [v_1 \cdots v_i \cdots v_n] \, .$$

De igual manera, dada una matriz $\mathbf{A} = [a_{ij}] \in \mathcal{R}_q^{n \times m}$, se define la matriz *transpuesta* de $\mathbf{A}$, y se denota $\mathbf{A}^T$ a la matriz que se obtiene al intercambiar las filas por las columnas, es decir, $\mathbf{A}^T = [a_{ji}] \in \mathcal{R}_q^{m \times n}$

Definición 2.1.3. Sea el cuerpo $\mathbb{K} = \mathbb{R}$ y $x \in \mathbb{R}$. Entonces, se define la función *valor absoluto* como la aplicación $|\cdot| : \mathbb{R} \longrightarrow \mathbb{R}^+ \cup \{0\}$ definida por

$$|x| = \begin{cases} x, & \text{si} \quad x \geq 0 \\ -x, & \text{si} \quad x < 0. \end{cases}$$

Esta definición de valor absoluto en $\mathbb{R}$ se puede generalizar en $\mathbb{C}$ de la siguiente forma:

Dado $z \in \mathbb{C}$, tal que $z = x + iy$ con $x, y \in \mathbb{R}$, definimos $|\cdot| : \mathbb{C} \longrightarrow \mathbb{R}^+ \cup \{0\}$ con

$$|z| = \sqrt{x^2 + y^2}$$

la cual adquiere el nombre de *módulo*.

Definición 2.1.4. Sea V un espacio vectorial sobre un cuerpo $\mathbb{K}$, con $\mathbb{K} = \mathbb{R}$ o $\mathbb{C}$. Entonces, una función $\|\cdot\| : V \longrightarrow \mathbb{K}$ se dice que es una *norma* si verifica:

i) Si $\mathbf{x} \neq 0 \Rightarrow \|\mathbf{x}\| > 0$ y $\mathbf{x} = 0 \Longleftrightarrow \|\mathbf{x}\| = 0,\ \forall\ \mathbf{x} \in V$

ii) $\|k\mathbf{x}\| = |k| \cdot \|\mathbf{x}\|, \forall\ \mathbf{x} \in V,\ k \in \mathbb{K}$

iii) $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|,\ \forall\ \mathbf{x}, \mathbf{y} \in V$ (Desigualdad triangular)

Definición 2.1.5. Sea V un $\mathbb{R}$ o $\mathbb{C}$ espacio vectorial. Dado el vector $\mathbf{x} = (x_1, \ldots, x_n) \in V$, se definen las *normas de Hölder* o *normas* ℓ_p como

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i| \right)^{1/p}$$

En el caso de la *norma infinito* se tiene

$$\|\mathbf{x}\|_\infty = \max_{1 \leq i \leq n} |x_i|$$

Es fácil comprobar que efectivamente son normas y que por tanto verifica la **Definición 2.1.4**.

Definición 2.1.6. Sea V un espacio vectorial sobre un cuerpo $\mathbb{K}$. Llamamos *producto escalar* a la función $\langle \cdot, \cdot \rangle : V \times V \longrightarrow \mathbb{K}$ que dados dos vectores $\mathbf{x}, \mathbf{y} \in V$ se tiene

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{n} y_i x_i = \mathbf{y}^T \mathbf{x}$$

Observación 2.1.1. Todo producto escalar induce una norma sobre el espacio en el que está definido, de la siguiente manera:

$$\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}$$

Definición 2.1.7. Dos vectores $\mathbf{x}, \mathbf{y} \in V$ se dice que son *ortogonales* si su producto escalar es cero:

$$\mathbf{x} \perp \mathbf{y} \Longleftrightarrow \langle \mathbf{x}, \mathbf{y} \rangle = 0$$

En el caso de las matrices, decimos que una matriz $\mathbf{A} \in \mathbb{R}^{n \times n}$ es *ortogonal* si verfica que $\mathbf{A} \cdot \mathbf{A}^T = \mathbf{A}^T \cdot \mathbf{A} = \mathbf{I}$.

Si $S \subseteq V$ es un subconjunto, definimos el *subespacio ortogonal de* S y lo denotamos

$$S^\perp = \left\{ \mathbf{y} \in V \;\middle|\; \langle \mathbf{x}, \mathbf{y} \rangle = 0, \forall\ \mathbf{x} \in S \right\}$$

Definición 2.1.8. Sean $\mathbf{b}_1, \ldots, \mathbf{b}_n$ vectores linealmente independientes en $\mathbb{R}^n$. Entonces, se definen *los vectores ortogonales de Gram-Schmidt* $\widetilde{\mathbf{b}}_1, \ldots, \widetilde{\mathbf{b}}_n$ como

$$\begin{cases} \widetilde{\mathbf{b}}_1 = & \mathbf{b}_1 \\ \widetilde{\mathbf{b}}_j = & \mathbf{b}_j - \sum_{i<j} \mu_{i,j} \widetilde{\mathbf{b}}_i \end{cases}$$

siendo $\mu_{i,j} = \dfrac{\langle \mathbf{b}_j, \widetilde{\mathbf{b}}_i \rangle}{\langle \widetilde{\mathbf{b}}_i, \widetilde{\mathbf{b}}_i \rangle}$ $\forall j \in \{2, \ldots, n\}$.

Además, como hemos dicho, los vectores $\widetilde{\mathbf{b}}_i$ son ortogonales entre ellos, es decir,

$$\langle \widetilde{\mathbf{b}}_i, \widetilde{\mathbf{b}}_j \rangle = 0 \quad \forall i \neq j$$

2.1.2. Retículos

Como habíamos introducido en el Prefacio, Crystals-Kyber es un criptosistema basado en retículos, por lo que vamos a ver las nociones básicas sobre estos:

Definición 2.1.9. Sea $\mathbb{K}$ un cuerpo y $\mathbf{b}_1, \ldots, \mathbf{b}_m$ vectores linealmente independientes en $\mathbb{K}^n$. Entonces, se define un *retículo*, $\mathcal{L}$, como el conjunto

$$\mathcal{L} \underset{not.}{\equiv} \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_m) = \left\{ \sum_{i=1}^{n} x_i \mathbf{b}_i \;\middle|\; x_i \in \mathbb{Z} \right\}$$

Además los vectores $\mathbf{b}_1, \ldots, \mathbf{b}_m$ forman una *base* del retículo $\mathcal{L}$ que podemos denotar como la matriz $\mathbf{B} = \left[\mathbf{b}_1, \ldots, \mathbf{b}_m\right] \in \mathbb{K}^{n \times m}$ lo que nos dejaría

$$\mathcal{L} \underset{not.}{\equiv} \mathcal{L}(\mathbf{B}) = \left\{ \mathbf{Bx} \;\middle|\; \mathbf{x} \in \mathbb{Z}^m \right\}$$

Si $n = m$, el *determinante* de un retículo es el valor absoluto del determinante de la matriz formada por los vectores de la base, es decir,

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

Proposición 2.1.2. *Si tomamos la matriz* $\mathbf{B}$ *formada por los vectores que forman la base del retículo* $\mathcal{L}$*, la podemos factorizar utilizando los vectores ortogonales de Gram-Schmidt, de la siguiente manera:*

$$\mathbf{B} = \widetilde{\mathbf{B}}\mathbf{U}$$

donde $\widetilde{\mathbf{B}} = [\widetilde{\mathbf{b}}_1, \ldots, \widetilde{\mathbf{b}}_m] \in \mathbb{K}^{n \times m}$ *y* $\mathbf{U} \in \mathbb{K}^{m \times m}$ *es la matriz triangular superior*

$$\mathbf{U} = \begin{bmatrix} 1 & \mu_{1,2} & \cdots & \mu_{1,m} \\ & 1 & \cdots & \mu_{2,m} \\ & & \ddots & \vdots \\ & & & 1 \end{bmatrix}$$

Además, podemos seguir factorizando la matriz $\widetilde{\mathbf{B}}$ *como* $\widetilde{\mathbf{B}} = \mathbf{QD}$ *donde* $\mathbf{Q} = \left[\frac{\widetilde{\mathbf{b}}_1}{\|\widetilde{\mathbf{b}}_1\|}, \ldots, \frac{\widetilde{\mathbf{b}}_m}{\|\widetilde{\mathbf{b}}_m\|}\right]$ *es una matriz ortogonal y*

$$\mathbf{D} = \begin{bmatrix} \|\widetilde{\mathbf{b}}_1\| & & & \\ & \|\widetilde{\mathbf{b}}_2\| & & \\ & & \ddots & \\ & & & \|\widetilde{\mathbf{b}}_m\| \end{bmatrix} \text{ es una matriz diagonal.}$$

De esta manera, nos quedaría una factorización única de B como

$$\mathbf{B} = \mathbf{QDU}$$

Esta factorización de $\mathbf{B}$, es un caso particular de la clásica *factorización QR* [24, Sección 6.4.], donde $\mathbf{R} = \mathbf{DU}$ es una matriz triangular superior, siendo $\|\mathbf{b}_i\|$ los elementos de la diagonal.

Definición 2.1.10. Sea $\mathcal{L}$ un retículo en $\mathbb{K}^n$. Se llama *dual* de $\mathcal{L}$ y se denota como $\mathcal{L}^*$ al conjunto formado por los vectores $\mathbf{y} \in \mathbb{F}^n$ que verifican que $\langle \mathbf{z}, \mathbf{y} \rangle \in \mathbb{Z}$ para todo vector $\mathbf{z} \in \mathcal{L}$. Es decir,

$$\mathcal{L}^* = \left\{ \mathbf{y} \in \mathbb{K}^n \middle| \langle \mathcal{L}, \mathbf{y} \rangle \subseteq \mathbb{Z} \right\}$$

Observación 2.1.2. Dada cualquier matriz $\mathbf{B} \in \mathbb{F}^{n \times n}$ formada por los vectores de una base del retículo $\mathcal{L}$, entonces como $\mathcal{L}(\mathbf{B})^* = \mathcal{L}((\mathbf{B}^{-1})^T)$ [21, Teorema 3.2.1.], concluimos que el determinante del dual es

$$\det(\mathcal{L}^*) = \frac{1}{\det(\mathcal{L})}$$

Definición 2.1.11. Sea $\mathcal{L}$ un retículo en $\mathbb{R}^n$ y q un número entero primo. Si tomamos una matriz $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, con $n \leq m$, se llama *retículo q-ario* a

$$\Lambda_q^{\perp}(\mathbf{A}) = \left\{ \mathbf{y} \in \mathbb{Z}^m \mid \mathbf{Ay} = \mathbf{0} \bmod q \right\}$$

tal que $\mathbb{Z}_q^m \subseteq \Lambda_q^{\perp}(\mathbf{A}) \subseteq \mathbb{Z}^m$.

De forma análoga podemos definir su dual como $(\Lambda_q^{\perp}(\mathbf{A}))^* = \frac{1}{q} \cdot \Lambda_q(\mathbf{A})$ siendo

$$\Lambda_q(\mathbf{A}) = \left\{ \mathbf{y} \in \mathbb{Z}^m \mid \mathbf{y} = \mathbf{A}^T\mathbf{s} \bmod q \text{ para algún } \mathbf{s} \in \mathbb{Z}^n \right\}.$$

2.2. Problema en el que está basado el esquema de Crystals-Kyber

Como bien hemos comentado con anterioridad, el esquema de Crystals-Kyber pertenece a la familia de los criptosistemas basados en retículos, por lo que centra su seguridad en el ya mencionado *Problema del Vector Más Corto (SVP)*. Por otro lado, el diseño de Kyber está basado en una versión del módulo del esquema de encriptación LWE, *Modulo-LWE*, que utiliza el problema Ring-LWE, siendo LWE las siglas del *Problema de Aprendizaje con Error*. En esta sección nos encargaremos de conocer mejor estos tipos de problemas y ver que ventajas e inconvenientes pueden presentar.

2.2.1. Problema del Vector Más Corto (SVP)

El problema se basa en que dado un retículo $\mathcal{L}$ y la matriz $\mathbf{B}$ base de $\mathcal{L}$, se quiere encontrar el vector no nulo más corto de $\mathcal{L}(\mathbf{B})$.

Se define como $\lambda_1(\mathcal{L})$ la *distancia mínima* de un retículo $\mathcal{L}$ a la longitud más corta de un vector del retículo, es decir, $\lambda_1(\mathcal{L}) = \min_{\mathbf{b}\in\mathcal{L}(\mathbf{B})} \|\mathbf{b}\|$. Además, gracias al teorema de Minkowski, sabemos cómo va a estar acotada esa distancia:

Teorema 2.2.1. ***(Teorema de Minkowski)*** *Sea $\mathcal{L}$ un retículo. Entonces existe un vector $\mathbf{x} \in \mathcal{L} \setminus \{0\}$ con $\|\mathbf{x}\| \leq \sqrt{n}\det(\mathcal{L})^{1/n}$.*

Demostración. Vease [4, Lecture 1] □

Ahora bien, recordemos que dados $\mathbf{b_1}, \ldots, \mathbf{b_n} \in \mathbb{R}^n$ vectores linealmente independientes podemos obtener los vectores ortogonales de Gram-Schmidt $\widetilde{\mathbf{b}}_1, \ldots, \widetilde{\mathbf{b}}_n$. Veamos que relación tienen estos vectores con cualquier retículo $\mathcal{L}$.

Lema 2.2.2. *Sea $\mathbf{B}$ la matriz formada por los vectores $\mathbf{b_1}, \ldots, \mathbf{b_n}$ de la base de $\mathcal{L}$. Entonces, se tiene que $\det(\mathcal{L}(\mathbf{B})) = \prod_{i=1}^{n} \|\widetilde{\mathbf{b}}_i\|$, donde $\widetilde{\mathbf{b}}_i$ $\forall i \in \{1, \ldots, n\}$ son los vectores ortogonales de Gram-Schmidt.*

Demostración. Recordemos que $\det(\mathcal{L}(\mathbf{B})) = |\det(\mathbf{B})|$ y que podemos escribir la matriz $\mathbf{B}$ como factorización única $\mathbf{B} = \mathbf{QDU}$, siendo $\mathbf{Q}$ una matriz ortogonal, $\mathbf{D}$ matriz diagonal con $\|\widetilde{\mathbf{b}}_i\|$ en la diagonal, y $\mathbf{U}$ matriz triangular superior con 1s en la diagonal. Así pues, y teniendo en cuenta que $\det(\mathbf{Q}) = 1$ por ser ortogonal y $\det(\mathbf{U}) = 1$ también por ser triangular superior y su diagonal 1s, se tiene

$$\det(\mathcal{L}(\mathbf{B})) = \det(\mathbf{B}) = \det(\mathbf{Q})\det(\mathbf{D})\det(\mathbf{U}) = \det(\mathbf{D}) = \prod_{i=1}^{n} \|\widetilde{\mathbf{b}}_i\|$$

□

Lema 2.2.3. *Sea $\mathbf{B}$ la matriz formada por los vectores $\mathbf{b_1}, \ldots, \mathbf{b_n}$ de la base de $\mathcal{L}$. Entonces, se tiene que $\lambda_1(\mathcal{L}(\mathbf{B})) = \min_i \|\widetilde{\mathbf{b}}_i\|$, donde $\widetilde{\mathbf{b}}_i$ $\forall i \in \{1, \ldots, n\}$ son los vectores ortogonales de Gram-Schmidt.*

Teniendo en cuenta estos últimos resultados, es decir, el **Teorema de Minkowski**, el **Lema 2.2.2** y el **Lema 2.2.3**, podemos deducir una acotación más precisa para la distancia mínima de un retículo $\mathcal{L}$:

$$\min_i \|\widetilde{\mathbf{b}}_i\| \leq \lambda_1(\mathcal{L}(\mathbf{B})) \leq \sqrt{n}\det(\mathcal{L}(\mathbf{B}))^{1/n} = \sqrt{n}\left(\prod_{i=1}^{n} \|\widetilde{\mathbf{b}}_i\|\right)^{1/n}$$

En el caso de los retículos q-arios, el problema se plantea como sigue:

Dada la matriz $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, con $n \leq m$, se desea encontrar el vector no nulo más corto en $\Lambda_q^{\perp}(\mathbf{A})$. Supongamos que el entero q es primo y que m no se encuentra muy cercano a n. Entonces, las filas de $\mathbf{A}$ serán linealmente independientes en $\mathbb{Z}_q$ y por ello, el número de elementos de $\mathbb{Z}_q^m$ que pertenecen a $\Lambda_q^{\perp}(\mathbf{A})$ es exactamente q^{m-n}. De esto se deduce que $\det(\Lambda_q^{\perp}(\mathbf{A})) = q^n$ y utilizando el Teorema de Minkowski obtendríamos que

$$\lambda_1(\Lambda_q^{\perp}(\mathbf{A})) \leq \sqrt{m}\det(\Lambda_q^{\perp}(\mathbf{A}))^{1/m} = \sqrt{m} \cdot q^{n/m}$$

2.2.2. Problema de Aprendizaje con Error (LWE)

Dados $n, m, q \in \mathbb{Z}$ este problema se basa en encontrar un vector $\mathbf{s} \in \mathbb{Z}_q^n$ tal que $\mathbf{As} + \mathbf{e} = \mathbf{b} \bmod q$ donde $\mathbf{A} \in \mathbb{Z}_q^{m \times n}, \mathbf{b} \in \mathbb{Z}_q^m$ y $\mathbf{e} \in \mathbb{Z}_q^m$. El vector $\mathbf{e}$ es el error y viene dado por la distribución χ^m de probabilidad en $\mathbb{Z}_q^m$. Teniendo todo esto en cuenta el problema LWE se expresa como el par $(\mathbf{A}, \mathbf{b})$.

Por otro lado, veamos a que hace referencia el problema *Ring-LWE* que comentabamos antes.

Tomamos el anillo $\mathcal{R}_q$. Entonces, el par $(\mathbf{A}, \mathbf{b})$ vendrá dado por $\mathbf{A} \in \mathcal{R}_q$ y $\mathbf{b} = \mathbf{As} + \mathbf{e} \bmod q\mathcal{R}$ con $\mathbf{s} \in \mathcal{R}_q$ y $\mathbf{e}$ obtenido de la distribución de probabilidad $\chi \bmod q$.

El diseño de Kyber se basa en la versión del módulo del esquema de encriptación Ring-LWE. Como acabamos de definir, en los esquemas Ring-LWE las operaciones son de la forma $\mathbf{As} + \mathbf{e}$ donde todas las variables son polinomios de algún anillo. La diferencia con lo utilizado en Crystals-Kyber es que la matriz $\mathbf{A}$ está descrita sobre un anillo polinómico $\mathcal{R}_q$ de tamaño constante, así como los vectores $\mathbf{s}, \mathbf{e}$ se encuentran también sobre el mismo anillo, que recordemos es $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$ con n siendo $2^{n'-1}$ tal que $(x^n + 1)$ es el $2^{n'}$-ésimo polinomio ciclotómico. Esto es el llamado *Modulo-LWE*.

La principal ventaja del esquema de encriptación Ring-LWE es la eficiencia, tanto en términos de velocidad, como del tamaño de la clave y 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.

Por otro lado, las ventajas del LWE estándar son la falta de estructura y la fácil escalabilidad, lo cual provoca que la eficiencia disminuya significa-

tivamente.

Modulo-LWE ofrece un equilibrio entre estos dos extremos. En el caso concreto de los parámetros de Modulo-LWE utilizados en Kyber, se ofrece una estructura algo reducida en comparación con la de Ring-LWE, pero una escalabilidad mucho mayor, y cuando se cifran mensajes de un tamaño fijo, un rendimiento muy similar a los esquemas basados en Ring-LWE.

2.3. Descripción del sistema y algoritmos

Todos los algoritmos descritos para Crystals-Kyber utilizan y devuelven el contenido en bytes, siendo $\mathcal{B}$ el conjunto $\{0, \ldots, 255\}$, $\mathcal{B}^k$ el conjunto de bytes de longitud k y $\mathcal{B}^*$ el conjunto de bytes de una longitud arbitraria. Además, denotaremos como $(a||b)$ a la concatenación de los bytes a y b.

Teniendo esto en cuenta, será necesario definir una función `BytesToBits` que pase un número l de bytes a uno de $8l$ bits, además de la funcion pseudoaleatoria $\texttt{PRF} : \mathcal{B}^{32} \times \mathcal{B} \to \mathcal{B}^*$, una función de salida extensible $\texttt{XOF} : \mathcal{B}^* \times \mathcal{B} \times \mathcal{B} \to \mathcal{B}^*$ y una función hash $\texttt{G} : \mathcal{B}^* \to \mathcal{B}^{32} \times \mathcal{B}^{32}$.

Por otro lado, necesitaremos las funciones $\texttt{Compress}_q(x, d)$ y $\texttt{Decompression}_q(x, d)$ definidas como

$$\begin{aligned} \texttt{Compress}_q(x, d) &= \left\lceil \tfrac{2^d}{q} x \right\rfloor \bmod 2^d \\ \texttt{Decompression}_q(x, d) &= \left\lceil \tfrac{q}{2^d} x \right\rfloor \end{aligned}$$

donde dado un $x \in \mathbb{Q}$, $\lceil x \rfloor$ es el *redondeo* de x al número entero más próximo y $d < \lceil \log_2 q \rceil$.

Por último, para realizar multiplicaciones en $\mathcal{R}_q$ utilizaremos la llamada *transformación teórica de números* (`NTT`), definida como la aplicación $\texttt{NTT} : \mathcal{R}_q \longrightarrow \mathcal{R}_q$ que dado $f = f_0 + f_1 X + \cdots + f_n X^n \in \mathcal{R}_q$ se tiene

$$\texttt{NTT}(f) = \hat{f}_0 + \hat{f}_1 X + \cdots \hat{f}_{n-1} X^{n-1} \underset{not.}{\equiv} \hat{f}$$

donde

$$\hat{f}_{2i} = \sum_{j=0}^{\frac{n}{2}-1} f_{2j} \omega^{(2\texttt{br}_7(i)+1)j}$$

$$\hat{f}_{2i+1} = \sum_{j=0}^{\frac{n}{2}-1} f_{2j+1} \omega^{(2\texttt{br}_7(i)+1)j}$$

siendo ω la primera de las n-raíces primitivas de la unidad, y $\texttt{br}_7(i)$ invierte los bits de un entero i de 7 bits.

Veamos ahora como son las funciones que sirven para generar las claves, encriptar un mensaje y desencriptarlo segun el esquema de Crystals-Kyber.

Los parametros utilizados en ellas son n, que tomará un valor fijo de 256, ya que el objetivo es encriptar textos de 256 bits de tamaño; q es un primo pequeño que debe satisfacer que $n|(q-1)$, que en este caso también tendrá un valor constante de $q = 3329$; $\omega = 17$, por ser la primera de las raíces primitivas para $n = 256$ módulo $q = 3329$; k se selecciona para fijar la dimensión de la red como un múltiplo de n por lo que nos marcará los diferentes niveles de seguridad; η_1, η_2, d_u y d_v se eligieron para equilibrar la seguridad, el tamaño del texto cifrado y la probabilidad de fallo. En el **Capítulo 3** mostraremos los valores que toman estos parametros dependiendo de los diferentes casos a analizar.

Empezamos por describir las funciones auxiliares que se van a utilizar:

Por un lado, tenemos la función CBD_η que obtiene una función pseudoaleatoria por distribución binomial centrada:

Algoritmo 1: $CBD_\eta(B)$

Entrada: Conjunto de bytes $B \in \mathcal{B}^{64\eta}$*, tamaño de*

Salida: Polinomio $f = f_0 + f_1X + \cdots + f_{n-1}X^{n-1}$

var $B \in \mathcal{B}^{64\eta}$; $a, b, \beta_i, f_i \in \{0,1\}$;$f \in \mathcal{R}_q$; $n, \eta \in \mathbb{Z}$ **end_var**
begin
`/* Pasamos cada elemento de` B `a listas de longitud 8 de 0s y 1` $\beta_i \dots \beta_{i+8}$ $\forall i \in \{0, \dots, 64\eta - 1\}$ `*/`
$(\beta_0, \dots, \beta_{512\eta-1}) \leftarrow \texttt{BytesToBits}(B)$
`/* Creación coeficientes del polinomio mediante bits */`
for i ***from*** 0 ***to*** $n-1$ **do**
  $a \leftarrow \sum_{j=0}^{\eta-1} \beta_{2i\eta+j}$
  $b \leftarrow \sum_{j=0}^{\eta-1} \beta_{2i\eta+\eta+j}$
  $f_i \leftarrow a - b$ `/* Coeficientes 0 o 1 */`
end
$f = f_0 + f_1X + \cdots + f_{n-1}X^{n-1}$
Return(f)
end

A continuación,la función $Parse : \mathcal{B}^* \to \mathcal{R}_q$ que se encarga de calcular la representación `NTT` de un conjunto de bytes B.

Algoritmo 2: *Parse*

Entrada: Conjunto de bytes $B \in \mathcal{B}^*$

Salida: Polinomio $\hat{a} = \hat{a}_0 + \hat{a}_1 X + \cdots + \hat{a}_{n-1} X^{n-1}$ *de representación NTT*

var $B \in \mathcal{B}^*$; $a \in \mathcal{R}_q$; $i, j, n, q, d_1, d_2 \in \mathbb{Z}$ **end_var**

begin
$i \leftarrow 0$
$j \leftarrow 0$
$n \leftarrow |B|$
while $j < n$ **do**
  $b_i \leftarrow B[i]$ `/* ` $B = (b_0, \ldots, b_{n-1})$ `*/`
  $d_1 \leftarrow b_i + n(b_{i+1} \bmod 16)$
  $d_2 \leftarrow \lfloor \frac{b_{i+1}}{16} \rfloor + 16 b_{i+2}$
  `/* Asignación coeficientes seleccionando el menor a` q `*/`
  if $d_1 < q$ **then**
    $\hat{a}a_j \leftarrow d_1$
    $j \leftarrow j + 1$
  end
  if $d_2 < q$ ***and*** $j < n$ **then**
    $\hat{a}_j \leftarrow d_2$
    $j \leftarrow j + 1$
  end
  $i \leftarrow i + 3$
end
$\hat{a} = \hat{a}_0 + \hat{a}_1 X + \cdots + \hat{a}_{n-1} X^{n-1}$
Return$(\hat{a})$
end

Por último, tenemos las funciones $Encode_\ell$ y $Decode_\ell$, que son las encargadas de pasar de un vector de polinomios a un conjunto de bytes y vicerversa, respectivamente. Como ambas funciones son una inversa de la otra, describimos el algoritmo de $Decode_\ell$:

Algoritmo 3: $Decode_\ell$

Entrada: *Conjunto de bytes* $B \in \mathcal{B}^{32\ell}$

Salida: *Polinomio* $f = f_0 + f_1X + \cdots + f_{n-1}X^{n-1}$

var $B \in \mathcal{B}^{32\ell}$; $f_i \in \{0, \ldots, 2^\ell - 1\}$; $f \in \mathcal{R}_q$; $n, \ell \in \mathbb{Z}$ **end_var**

begin
`/* Pasamos cada elemento de` B `a listas de longitud 8 de 0s y 1` $\beta_i \ldots \beta_{i+8}$ $\forall i \in \{0, \ldots, 64\eta - 1\}$ `*/`
$(\beta_0, \ldots, \beta_{n\ell-1}) \leftarrow$ `BytesToBits`(B)
`/* Creación coeficientes del polinomio mediante bits */`
for i ***from*** 0 ***to*** $n - 1$ **do**
  $f_i \leftarrow \sum_{j=0}^{\ell-1} \beta_{i\ell+j} 2^j$ `/* Coeficientes del 0 al` $2^\ell - 1$ `*/`
end
$f = f_0 + f_1X + \cdots + f_{n-1}X^{n-1}$
Return(f)
end

Pasamos ahora a presentar los algoritmos principales, empezando por el de la generación de clave, que nos devolvera una clave pública k_{pub} y una clave privada k_{priv} escrita en bits:

Algoritmo 4: $KeyGen()$

Salida: Clave pública k_{pub}, clave privada k_{priv}

var $d, \rho, \sigma, k_{pub}, k_{priv} \in \mathcal{B}^*$; $\hat{\mathbf{A}} \in \mathcal{R}_q^{k \times k}$; $\mathbf{s}, \hat{\mathbf{s}}, \mathbf{e}, \hat{\mathbf{e}}, \mathbf{b} \in \mathcal{R}_q^k$; $N, k, \eta_1 \in \mathbb{Z}$
end_var

begin
$d \leftarrow \mathcal{B}^{32}$ `/* Cálculo aleatorio de bytes de longitud 32 */`
$(\rho, \sigma) \leftarrow \mathtt{G}(d)$
$N \leftarrow 0$
`/* Creación de la matriz` $\hat{\mathbf{A}} \in \mathcal{R}_q^{k \times k}$ `siendo los coeficientes polinomios de representación NTT */`
for i ***from*** 0 ***to*** $k-1$ **do**
  for j ***from*** 0 ***to*** $k-1$ **do**
    $\hat{\mathbf{A}}[i][j] \leftarrow Parse(\mathtt{XOF}(\rho, j, i))$
  end
end
`/* Creación del vector` $\mathbf{s} \in \mathcal{R}_q^k$ `partiendo de` B_{η_1} `*/`
for i ***from*** 0 ***to*** $k-1$ **do**
  $\mathbf{s}[i] \leftarrow CBD_{\eta_1}(\mathtt{PRF}(\sigma, N))$ `/* Coeficientes con creación aleatoria de bits */`
  $N \leftarrow N + 1$
end
`/* Creación del vector` $\mathbf{e} \in \mathcal{R}_q^k$ `partiendo de` B_{η_1} `*/`
for i ***from*** 0 ***to*** $k-1$ **do**
  $\mathbf{e}[i] \leftarrow CBD_{\eta_1}(\mathtt{PRF}(\sigma, N))$ `/* Coeficientes con creación aleatoria de bits */`
  $N \leftarrow N + 1$
end
`/* Realizar las operaciones como se definen en NTT */`
$\hat{\mathbf{s}} \leftarrow \mathtt{NTT}(\mathbf{s})$
$\hat{\mathbf{e}} \leftarrow \mathtt{NTT}(\mathbf{e})$
$\hat{\mathbf{b}} \leftarrow \hat{\mathbf{A}}\hat{\mathbf{s}} + \hat{\mathbf{e}}$
`/* Usamos` $Encode_\ell : \mathcal{R}_q \longrightarrow \mathcal{B}^*$ `para devolver en bits */`
$k_{pub} \leftarrow Encode_{12}(\hat{\mathbf{b}} \bmod q)$ `/* Transformación` $\hat{\mathbf{b}}$ `a bits */`
$k_{priv} \leftarrow Encode_{12}(\hat{\mathbf{s}} \bmod q)$ `/* Transformación` $\hat{\mathbf{s}}$ `a bits */`
Return(k_{pub}, k_{priv})
end

Ahora veamos el algoritmo encargado de cifrar un mensaje m, utilizando la clave pública k_{pub}, para devolverlo como en texto cifrado c, que así como m, vendrá expresado en bits:

Algoritmo 5: *Encryption*(k_{pub}, m, s)

Entrada: Clave pública k_{pub}, mensaje m, bytes aleatorios s

Salida: Texto cifrado c

var $s, \rho, k_{pub}, m, c_1, c_2, c \in \mathcal{B}^*$; $\hat{\mathbf{A}} \in \mathcal{R}_q^{k \times k}$; $\mathbf{s}, \hat{\mathbf{s}}, \mathbf{e_1}, \mathbf{b}, \hat{\mathbf{b}} \in \mathcal{R}_q^k$; $e_2 \in \mathcal{R}_q$
$N, k, n, q, d_u, d_v \in \mathbb{Z}$ **end_var**

begin
$N \leftarrow 0$
$\mathbf{b} \leftarrow Decode_{12}(k_{pub})$ `/* Transformación` k_{pub} `a` $\mathcal{R}_q$ `*/`
$\rho \leftarrow k_{pub} + 12k\frac{n}{8}$
`/* Creación de la matriz` $\hat{\mathbf{A}} \in \mathcal{R}_q^{k \times k}$ `siendo los coeficientes polinomios de representación NTT */`
for i ***from*** 0 ***to*** $k-1$ **do**
  for j ***from*** 0 ***to*** $k-1$ **do**
    $\hat{\mathbf{A}}[i][j] \leftarrow Parse(\texttt{XOF}(\rho, j, i))$
  end
end
`/* Creación del vector` $\mathbf{s} \in \mathcal{R}_q^k$ `partiendo de` B_{η_1} `*/`
for i ***from*** 0 ***to*** $k-1$ **do**
  $\mathbf{s}[i] \leftarrow CBD_{\eta_1}(\texttt{PRF}(s, N))$
  $N \leftarrow N + 1$
end
`/* Creación del vector` $\mathbf{e_1} \in \mathcal{R}_q^k$ `partiendo de` B_{η_2} `*/`
for i ***from*** 0 ***to*** $k-1$ **do**
  $\mathbf{e}_1[i] \leftarrow CBD_{\eta_2}(\texttt{PRF}(s, N))$
  $N \leftarrow N + 1$
end
`/* Creación del coeficiente` $e_2 \in \mathcal{R}_q$ `partiendo de` B_{η_2} `*/`
$e_2 \leftarrow CBD_{\eta}(\texttt{PRF}(s, N))$
`/* Realizar las operaciones como se definen en NTT */`
$\hat{\mathbf{s}} \leftarrow \texttt{NTT}(\mathbf{s})$
$\mathbf{u} \leftarrow \texttt{NTT}^{-1}(\hat{\mathbf{A}}^T \hat{\mathbf{s}}) + \mathbf{e}_1$
$\mathbf{v} \leftarrow \texttt{NTT}^{-1}(\hat{\mathbf{b}}^T \hat{\mathbf{s}}) + \mathbf{e}_2 + \texttt{Decompression}_q(Decode_1(m), 1)$
$c_1 \leftarrow Encode_{d_u}(\texttt{Compress}_q)(\mathbf{u}, d_u)$
$c_2 \leftarrow Encode_{d_v}(\texttt{Compress}_q)(\mathbf{v}, d_v)$ $c \leftarrow (c_1 || c_2)$
Return(c)
end

Por último, tenemos la descripción del algoritmo encargado de que dado un texto cifrado c, lo convierta en un mensaje claro m, dados ambos en bits, para lo cual utilizará, como es habitual en estos criptosistemas, la clave privada k_{priv}:

Algoritmo 6: *Decryption*(k_{priv}, c)

Entrada: Clave privada k_{priv}, texto cifrado c

Salida: Mensaje claro m

var $k_{priv}, c, m \in \mathcal{B}^*$; $\mathbf{u}, \mathbf{v}, \hat{\mathbf{s}} \in \mathcal{R}_q^k$; $k, n, q, d_u, d_v \in \mathbb{Z}$ **end_var**

begin
$\mathbf{u} \leftarrow \texttt{Decompression}_q(Decode_{d_u}(c), d_u)$
$\mathbf{v} \leftarrow \texttt{Decompression}_q(Decode_{d_v}(c + d_u k\frac{n}{8}), d_v)$
$\hat{\mathbf{s}} \leftarrow Decode_{12}(k_{priv})$
`/* Realizar las operaciones` $\mathbf{v} - \hat{\mathbf{s}}^T\mathbf{u}$ `como se definen en NTT y transformar a bits */`
$m \leftarrow Encode_1(\texttt{Compress}_q(\mathbf{v} - \texttt{NTT}^{-1}(\hat{\mathbf{s}}^T\texttt{NTT}(\mathbf{u})), 1))$
Return(m)
end

Capítulo 3

Análisis y comparativa

En este último capítulo, vamos a analizar los tiempos de ejecución de los modelos propuestos en el esquema de *CRYSTALS-KYBER* para la generación de clave, la encriptación y la desencriptación. Por otro lado, vamos a realizar ese mismo análisis con el resto de esquemas clasificados para la tercera ronda, para así poder compararlos entre ellos.

3.1. Descripción del proceso

Para la realización de este análisis, hemos utilizado una librería diseñada para el proyecto de *Open Quantum Safe (OQS)*, proyecto creado para apoyar el desarrollo y la creación de prototipos resistentes a la criptografía cuántica.

La librería, llamada `liboqs`, es una librería de código abierto en `C` que dispone de implementaciones de 9 KEMs (los 4 finalistas del proceso del NIST, así como 5 candidatos alternativos) y 5 esquemas de firma (3 finalistas y 2 alternativos). Debemos destacar que la versión utilizada para este trabajo ha sido la **versión 0.5.0** (10 de marzo de 2021), siendo a día de la redacción de esta memoria, la versión más reciente.

Por otro lado, el OQS dispone de unos "wrappers"para utilizar los algoritmos postcuánticos de `liboqs` en diferentes lenguajes de programación. Nosotros hemos utilizado el de `Python` con **versión 0.4.0** (24 de marzo de 2021).

Por último, es necesario mencionar que el ordenador utilizado para realizar todas las pruebas que mostraremos en las siguientes secciones dispone de un procesador **AMS A10-4655M APU with Radeon(tm) Graphics 2.00 GHz**, una memoria RAM **DDR3L** de **4.00 GB** y un sistema operativo **Windows 10 Home de 64 bits con procesador basado en x64**.

3.2. Resultados de Crystals-Kyber

El esquema de Crystals-Kyber cuenta con un total de 6 modelos diferentes; 3 originales **Kyber512, Kyber768, Kyber1024** y una variante de cada uno de ellos, llamados **Kyber512-90s, Kyber768-90s, Kyber1024-90s**. La diferencia entre los originales y las variantes se debe al cálculo de las funciones `PRF, XOF, G` que coge diferentes modelos del estándar FIPS-202 [15]:

- *Originales*:
  - `PRF` $\longrightarrow$ SHAKE-256
  - `XFO` $\longrightarrow$ SHAKE-128
  - `G` $\longrightarrow$ SHA3-512
- *Variantes* $-$ 90*s*:
  - `PRF` $\longrightarrow$ AES-256
  - `XFO` $\longrightarrow$ AES-256
  - `G` $\longrightarrow$ SHA-512

Por otro lado, utilizando parte del **Programa(A.5)**, hemos creado la siguiente tabla donde se explican las diferentes caracteristicas de cada modelo:

Datos	**Kyber512**	**Kyber768**	**Kyber1024**
Nivel de seguridad NIST reclamado	1	3	5
Tamaño del texto cifrado (bytes)	768	1088	1568
Tamaño de clave pública (bytes)	800	1184	1568
Tamaño de clave privada (bytes)	1632	2400	3168
Datos	**Kyber512-90s**	**Kyber768-90s**	**Kyber1024-90s**
Nivel de seguridad NIST reclamado	1	3	5
Tamaño del texto cifrado (bytes)	768	1088	1568
Tamaño de clave pública (bytes)	800	1184	1568
Tamaño de clave privada (bytes)	1632	2400	3168

Tabla 3.1: Datos del esquema de Crystals-Kyber

Por último, puesto que se han utilizado los algoritmos de generación de clave, encriptado y desencriptado, mencionadas en el **Capitulo 2**, es necesario destacar qué parámetros se utilizan para cada uno de los modelos:

	n	**k**	**q**	η_1	η_2	(d_u, d_v)
Kyber512(-90s)	256	2	3329	3	2	(10, 4)
Kyber768(-90s)	256	3	3329	2	2	(10, 4)
Kyber1024(-90s)	256	4	3329	2	2	(11, 5)

Tabla 3.2: Conjuntos de parámetros para Crystals-Kyber

3.2.1. Generación de clave

En el siguiente gráfico podemos observar los tiempos obtenidos de los 6 modelos. Se puede apreciar que a medida que aumentan los tamaños de clave, tanto la parte privada como la parte pública, así como los bits del mensaje cifrado, el tiempo de ejecución es mayor, tal y como se esperaba. Además, podemos ver que son más rápidas las versiones originales en comparación con sus respectivas variantes, incluso podemos destacar que en el caso de **Kyber1024** y **Kyber768-90s** la diferencia es mínima y la variante utiliza tamaños mucho menores.

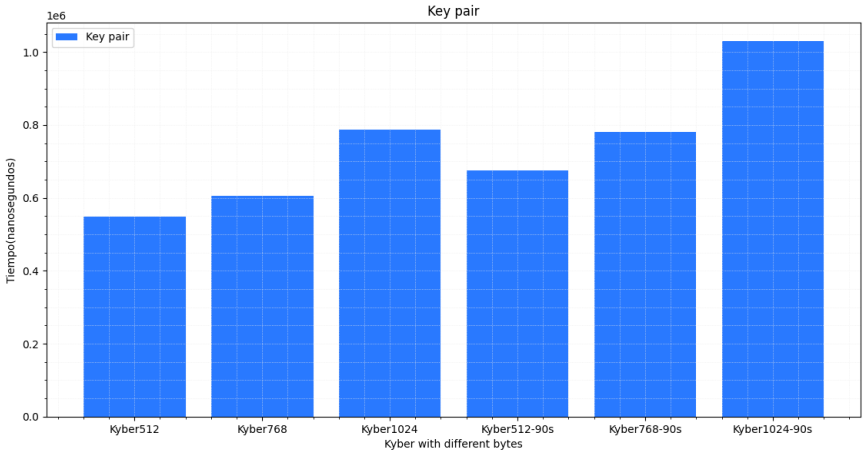

Figura 3.1: Gráfica diferentes modelos de Kyber generando clave

3.2.2. Cifrado del mensaje

Para el caso del cifrado podemos observar algo bastante similar a la generación de claves, aunque cabe destacar que el tiempo de ejecución es bastante más inferior que el anterior.

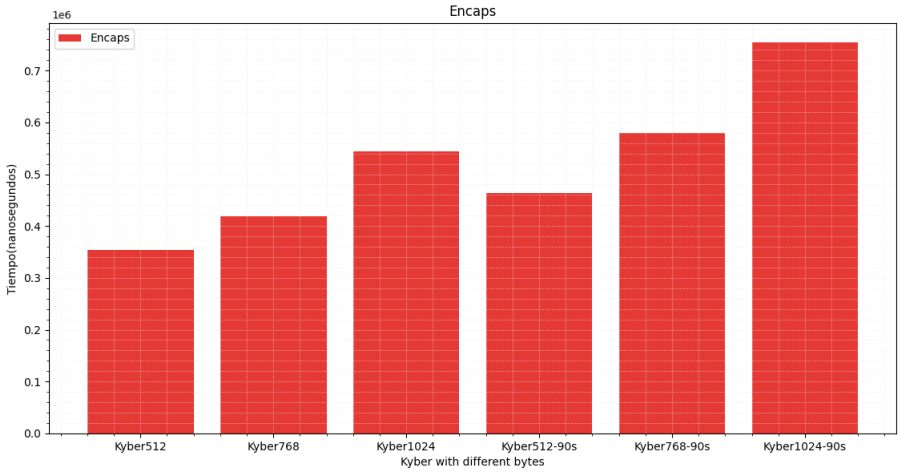

Figura 3.2: Gráfica diferentes modelos de Kyber encriptando

3.2.3. Descifrado del mensaje

Por último, a la hora de descifrar un mensaje, podemos apreciar una estructura parecida a los dos casos anteriores, concluyendo así lo esperado, que a mayor nivel de seguridad, es decir, mayor tamaño de longitud en los bits utilizados para las claves y los mensajes, es necesario un mayor tiempo de espera. Aun así, la diferencia se podría decir que no es tan significativa como para que pudiera perder eficacia.

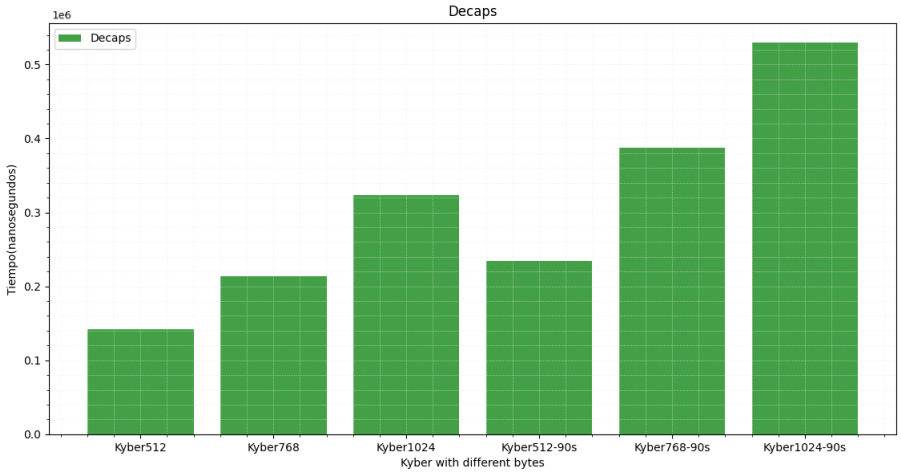

Figura 3.3: Gráfica diferentes modelos de Kyber desencriptando

3.3. Resultados de los finalistas y comparativa

La idea inicial era hacer una comparativa entre los 4 esquemas finalistas, pero se ha tenido que descartar el esquema de *ClassisMcEliece* dado que se generaban errores a la hora de ejecutarlo, lo cual era debido a los grandes tamaños de bits que utiliza. De igual manera que para el esquema de *Kyber*, hemos representado en las siguientes tablas los datos de los esquemas *NTRU* y *Saber*:

Datos	**NTRU-HPS-2048-509**	**NTRU-HPS-2048-677**
Nivel de seguridad NIST reclamado	1	3
Tamaño del texto cifrado (bytes)	699	930
Tamaño de clave pública (bytes)	699	930
Tamaño de clave privada (bytes)	935	1234
Datos	**NTRU-HPS-4096-821**	**NTRU-HRSS-701**
Nivel de seguridad NIST reclamado	5	3
Tamaño del texto cifrado (bytes)	1230	1138
Tamaño de clave pública (bytes)	1230	1138
Tamaño de clave privada (bytes)	1590	1450

Tabla 3.3: Datos del esquema de NTRU

Datos	**LightSaber-KEM**	**Saber-KEM**	**FireSaber-KEM**
Nivel de seguridad NIST reclamado	1	3	5
Tamaño del texto cifrado (bytes)	736	1088	1472
Tamaño de clave pública (bytes)	672	992	1312
Tamaño de clave privada (bytes)	1568	2304	3040

Tabla 3.4: Datos del esquema de Saber

3.3.1. Generación de clave

Obtenida la siguiente gráfica, en la que podemos comparar de forma visual los tiempos de ejecución en la generación de clave para los 3 finalistas seleccionados, es llamativa la gran diferencia que muestra *NTRU* con respecto al resto, siendo su tiempo superior. En cambio no se aprecian grandes diferencias entre *Kyber* y *Saber*, siendo por tanto ambos, bastante eficientes.

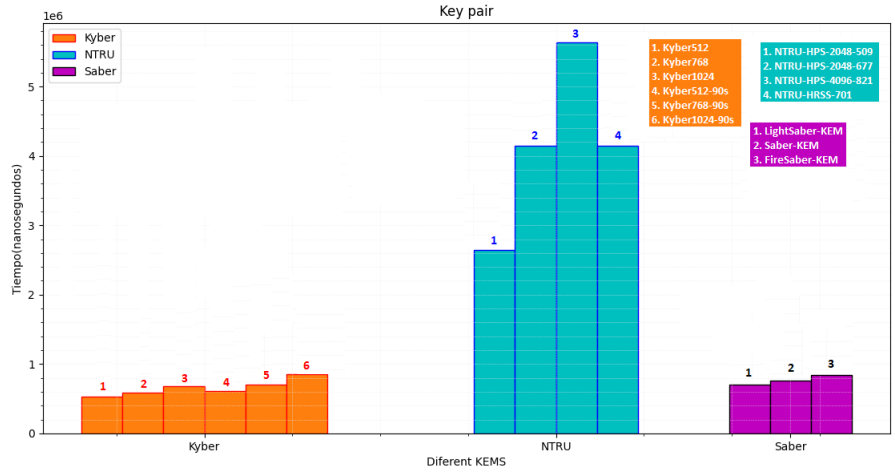

Figura 3.4: Gráfica comparativa de generación de clave entre finalistas

3.3.2. Cifrado del mensaje

En este caso, llama la atención nuevamente el esquema *NTRU*. Podemos observar que el tiempo de ejecución en la encriptación de un mensaje con un nivel de seguridad 3, referente a **NTRU-HRSS-701**, es inferior a aquellos que tienen tamaños de longitud menor.

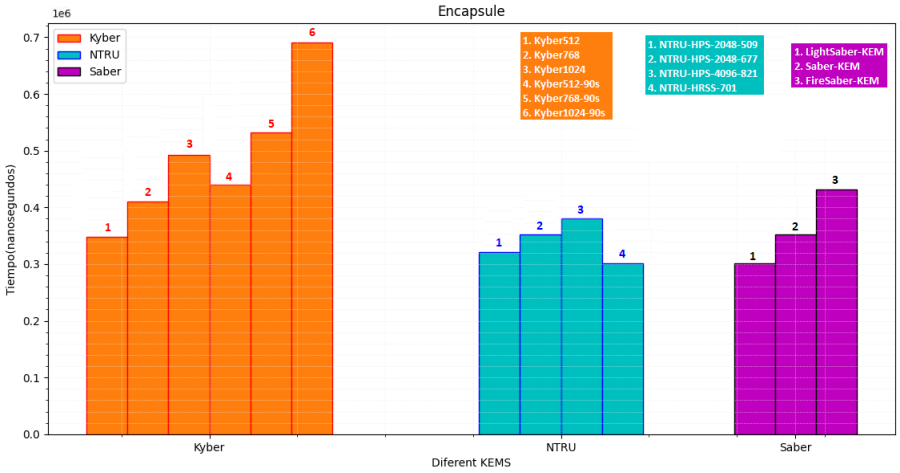

Figura 3.5: Gráfica comparativa de encriptación entre finalistas

3.3.3. Descifrado del mensaje

En cuanto al descifrado, ocurre algo muy similar al cifrado del mensaje por lo que al comparar unos con otros, podemos observar que destaca con un mayor tiempo *Kyber*, pero esa diferencia se podría deber a que utiliza tamaños ligeramente mayores.

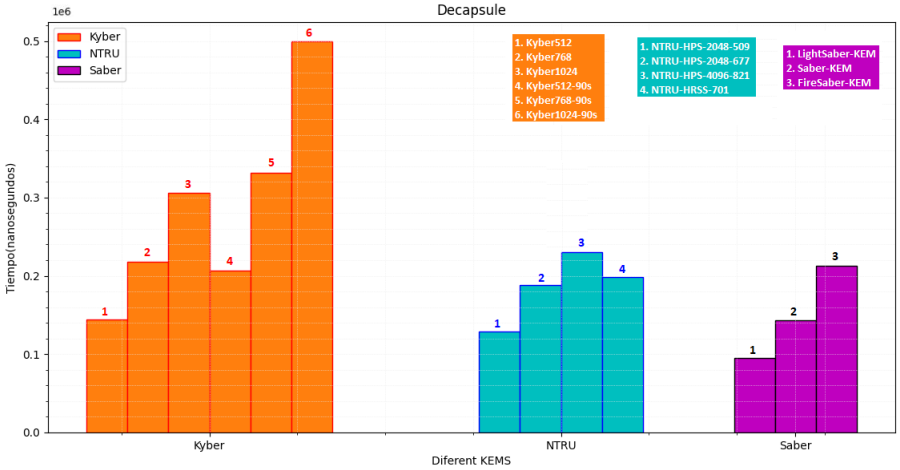

Figura 3.6: Gráfica comparativa de desencriptación entre finalistas

3.3.4. Conclusión de la comparativa

Una vez analizados los tres finalistas, y aunque debemos tener en cuenta a la hora de comparar unos con otros que los tamaños utilizados no son exactamente los mismos, podemos apreciar que el sistema criptográfico *Saber* tiene muy buenos tiempos de ejecución en todas las etapas analizadas. Además podemos destacar con una mayor claridad al verlo de forma conjunta, que las variantes utilizadas en *Kyber* no son nada buenas, ya que no solo incrementa el tiempo con respecto a las originales si no que la diferencia con los modelos de *NTRU* y *Saber* de tamaños similares es muy notable.

Apéndice A

Programación

En esta sección, se incluyen los programas diseñados, en el lenguaje de programación de `Python`, para la realización de los calculos de algunos de los ejemplos expuestos en el **Capítulo 1**, así como para la implementación de los gráficos introducidos en el **Capítulo 3**.

Programa A.1. Programa diseñado para calcular el punto que se obtiene como resultado de la suma de dos puntos diferentes, tal y como se ha definido en la **Sección 1.2.3**.

```
def PmasQ(P,Q,p):
    R =list()
    (xp,yp) = P
    (xq,yq) = Q
    r = pow(xq-xp,-1,p)
    s = (yq-yp)*r
    R.append(pow(s**2-xq-xp,1,p))
    R.append(pow(s*(xp-R[0])-yp,1,p))
    return R
```

Programa A.2. Programa diseñado para realizar la suma de un punto con él mismo, tal y como se ha descrito en la **Sección 1.2.3**.

```
def PmasP(P,a,p):
    R = list()
    (x,y) = P
    r = pow(2*y,-1,p)
    s = (3*x**2+a)*r
    R.append(pow(s**2-2*x,1,p))
    R.append(pow(s*(x-R[0])-y,1,p))
    return R
```

Programa A.3. Programa diseñado para calcular el punto obtenido como la suma de un punto de una curva elíptica, durante un número finito de veces.

```
def nP(n,P,a,p):
    R = P
    for i in range(1,n):
        if i == 1:
            R = PmasP(R,a,p)
        else:
            R = PmasQ(R,P,p)
    return R
```

Programa A.4. Programa, que utiliza la librería `liboqs`, diseñado para realizar una comparación de los diferentes modelos de Kyber de los que dispone la librería, frente a sus tiempos de ejecución durante la creación de claves, encriptación y desencriptación. Dicha comparación vendrá representada a traves de un gráfico de barras.

```
from pprint import pprint
import oqs
from time import *
import numpy as np
import matplotlib.pyplot as plt

##############################################################
#####################  KYBER COMPARATION  ####################
##############################################################

kems = oqs.get_enabled_KEM_mechanisms()

kemalgs = ['Kyber512', 'Kyber768', 'Kyber1024',
        'Kyber512-90s', 'Kyber768-90s', 'Kyber1024-90s']
tiempo_key = list()
tiempo_encaps = list()
tiempo_decaps = list()
for kemalg in kemalgs:
    with oqs.KeyEncapsulation(kemalg) as client:
        with oqs.KeyEncapsulation(kemalg) as server:
            print("\nKey encapsulation details:")
            pprint(client.details)

            # client generates its keypair
            tiempo = 0
```

```
            for i in range(100000):
                time1 = time_ns()
                public_key = client.generate_keypair()
                secret_key = client.export_secret_key()
                time2 = time_ns()
                tiempo = tiempo + (time2-time1)/100000
            tiempo_key.append(tiempo)

            # the server encapsulates its secret
            # using the client's public key
            tiempo = 0
            for i in range(100000):
                time1 = time_ns()
                ciphertext, shared_secret_server =
                    server.encap_secret(public_key)
                time2 = time_ns()
                tiempo = tiempo + (time2-time1)/100000
            tiempo_encaps.append(tiempo)

            # the client decapsulates the server's
            # ciphertext to obtain the shared secret
            tiempo = 0
            for i in range(100000):
                time1 = time_ns()
                shared_secret_client =
                    client.decap_secret(ciphertext)
                time2 = time_ns()
                tiempo = tiempo + (time2-time1)/100000
            tiempo_decaps.append(tiempo)
# Graficas

# Key
plt.figure(1)
plt.bar(kemalgs, tiempo_key, label = 'Key pair',
        color = '#2979ff')
plt.xlabel('Kyber with different bytes')
plt.ylabel('Tiempo(nanosegundos)')
plt.title('Key pair')
plt.legend(loc='upper left')
plt.grid(which='major', color='#EEEEEE', linestyle=':',
        linewidth=0.5)
plt.grid(which='minor', color='#EEEEEE', linestyle=':',
        linewidth=0.5)
plt.minorticks_on()
```

```
# Encaps
plt.figure(2)
plt.bar(kemalgs, tiempo_encaps, label = 'Encaps',
        color = '#e53935')
plt.xlabel('Kyber with different bytes')
plt.ylabel('Tiempo(nanosegundos)')
plt.title('Encaps')
plt.legend(loc='upper left')
plt.grid(which='major', color='#EEEEEE', linestyle=':',
        linewidth=0.5)
plt.grid(which='minor', color='#EEEEEE', linestyle=':',
        linewidth=0.5)
plt.minorticks_on()

# Decaps
plt.figure(3)
plt.bar(kemalgs, tiempo_decaps, label = 'Decaps',
        color = '#43a047')
plt.xlabel('Kyber with different bytes')
plt.ylabel('Tiempo(nanosegundos)')
plt.title('Decaps')
plt.legend(loc='upper left')
plt.grid(which='major', color='#EEEEEE', linestyle=':',
        linewidth=0.5)
plt.grid(which='minor', color='#EEEEEE', linestyle=':',
        linewidth=0.5)
plt.minorticks_on()

plt.show()
```

Programa A.5. Programa, que utiliza la librería `liboqs`, diseñado para realizar una comparación de tres de los finalistas del NIST, utilizando de cada uno de ellos los diferentes modelos de los que dispone la librería, frente a sus tiempos de ejecución durante la creación de claves, encriptación y desencriptación. Dicha comparación vendrá representada a traves de un gráfico de barras. Además, también crea una tabla con las características de dichos modelos en formato LaTeX.

```
from pprint import pprint
import oqs
from time import *
import numpy as np
```

```
import matplotlib.pyplot as plt
from tabulate import tabulate

############################################################
################ ROUND 3 KEMS CONPARATION ##################
############################################################

kems = oqs.get_enabled_KEM_mechanisms()
Kyber = ['Kyber512', 'Kyber768', 'Kyber1024',
         'Kyber512-90s', 'Kyber768-90s', 'Kyber1024-90s']
NTRU = ['NTRU-HPS-2048-509',       'NTRU-HPS-2048-677',
         'NTRU-HPS-4096-821', 'NTRU-HRSS-701']
Saber = ['LightSaber-KEM', 'Saber-KEM', 'FireSaber-KEM']

Kyber_key = list()
Kyber_encaps = list()
Kyber_decaps = list()

NTRU_key = list()
NTRU_encaps = list()
NTRU_decaps = list()

Saber_key = list()
Saber_encaps = list()
Saber_decaps = list()


###################### KYBER ############################

ld_Kyber = list()

for kemalg in Kyber:
    with oqs.KeyEncapsulation(kemalg) as client:
        with oqs.KeyEncapsulation(kemalg) as server:
            ld_Kyber.append(client.details)

            # client generates its keypair
            tiempo = 0
            for i in range(100000):
                time1 = time_ns()
                public_key = client.generate_keypair()
                secret_key = client.export_secret_key()
                time2 = time_ns()
                tiempo = tiempo + (time2-time1)/100000
```

```
            Kyber_key.append(tiempo)

            # the server encapsulates its secret using
            # the client's public key
            tiempo = 0
            for i in range(100000):
                time1 = time_ns()
                ciphertext, shared_secret_server =
                    server.encap_secret(public_key)
                time2 = time_ns()
                tiempo = tiempo + (time2-time1)/100000
            Kyber_encaps.append(tiempo)

            # the client decapsulates the the server's
            # ciphertext to obtain the shared secret
            tiempo = 0
            for i in range(100000):
                time1 = time_ns()
                shared_secret_client =
                    client.decap_secret(ciphertext)
                time2 = time_ns()
                tiempo = tiempo + (time2-time1)/100000
            Kyber_decaps.append(tiempo)


##################### NTRU ############################

ld_NTRU = list()

for kemalg in NTRU:
    with oqs.KeyEncapsulation(kemalg) as client:
        with oqs.KeyEncapsulation(kemalg) as server:
            ld_NTRU.append(client.details)

            # client generates its keypair
            tiempo = 0
            for i in range(100000):
                time1 = time_ns()
                public_key = client.generate_keypair()
                secret_key = client.export_secret_key()
                time2 = time_ns()
                tiempo = tiempo + (time2-time1)/100000
            NTRU_key.append(tiempo)
```

```
            # the server encapsulates its secret using
            # the client's public key
            tiempo = 0
            for i in range(100000):
                time1 = time_ns()
                ciphertext, shared_secret_server =
                    server.encap_secret(public_key)
                time2 = time_ns()
                tiempo = tiempo + (time2-time1)/100000
            NTRU_encaps.append(tiempo)

            # the client decapsulates the the server's
            # ciphertext to obtain the shared secret
            tiempo = 0
            for i in range(100000):
                time1 = time_ns()
                shared_secret_client =
                    client.decap_secret(ciphertext)
                time2 = time_ns()
                tiempo = tiempo + (time2-time1)/100000
            NTRU_decaps.append(tiempo)


########################## SABER ##############################

ld_Saber = list()

for kemalg in Saber:
    with oqs.KeyEncapsulation(kemalg) as client:
        with oqs.KeyEncapsulation(kemalg) as server:
            ld_Saber.append(client.details)

            # client generates its keypair
            tiempo = 0
            for i in range(100000):
                time1 = time_ns()
                public_key = client.generate_keypair()
                secret_key = client.export_secret_key()
                time2 = time_ns()
                tiempo = tiempo + (time2-time1)/100000
            Saber_key.append(tiempo)

            # the server encapsulates its secret using
            # the client's public key
```

```
            tiempo = 0
            for i in range(100000):
                time1 = time_ns()
                ciphertext, shared_secret_server =
                    server.encap_secret(public_key)
                time2 = time_ns()
                tiempo = tiempo + (time2-time1)/100000
            Saber_encaps.append(tiempo)

            # the client decapsulates the the server's
            # ciphertext to obtain the shared secret
            tiempo = 0
            for i in range(100000):
                time1 = time_ns()
                shared_secret_client =
                    client.decap_secret(ciphertext)
                time2 = time_ns()
                tiempo = tiempo + (time2-time1)/100000
            Saber_decaps.append(tiempo)


#### CREACION DE TABLA CON LOS DATOS ####

###################### KYBER ############################
d_Kyber = dict()
d_Kyber['Datos'] = d_Kyber512.keys()
for d in ld_Kyber:
    del d['version'] # Eliminamos el link de la version
    del d['is_ind_cca'] # Eliminamos 'is_ind_cca'
    nombre = d.get('name')
    # Eliminamos el nombre ya que lo pondremos de cabecera
    del d['name']
    d_Kyber[nombre] = d.values()

print(tabulate(d_Kyber, headers = 'keys', tablefmt =
        'latex_booktabs', stralign = 'center'))

###################### NTRU ############################
d_NTRU = dict()
d_NTRU['Datos'] = d_NTRU_HPS_2048_509.keys()
for d in ld_NTRU:
    del d['version'] # Eliminamos el link de la version
    del d['is_ind_cca'] # Eliminamos 'is_ind_cca'
    nombre = d.get('name')
```

```
    # Eliminamos el nombre ya que lo pondremos de cabecera
    del d['name']
    d_NTRU[nombre] = d.values()

print(tabulate(d_NTRU, headers = 'keys', tablefmt =
        'latex_booktabs', stralign = 'center'))

##################### SABER ############################
d_Saber = dict()
d_Saber['Datos'] = d_LightSaber_KEM.keys()
for d in ld_Saber:
    del d['version'] # Eliminamos el link de la version
    del d['is_ind_cca'] # Eliminamos 'is_ind_cca'
    nombre = d.get('name')
    # Eliminamos el nombre ya que lo pondremos de cabecera
    del d['name']
    d_Saber[nombre] = d.values()

print(tabulate(d_Saber, headers = 'keys', tablefmt =
        'latex_booktabs', stralign = 'center'))

#### GRAFICAS ####

labels = ["Kyber", "NTRU","Saber"]
x = np.arange(len(labels))
x[1] = x[0] + len(Kyber)//2
x[2] = x[1] + len(NTRU)//2
width = 0.35  # the width of the bars
y = list()
if len(Kyber)%2 == 0: # longitud par
    y.append(x[0]-(len(Kyber)//2 -1)*width - width/2)
else:
    y.append(x[0]-(len(Kyber)//2)*width)

if len(NTRU)%2 == 0: # longitud par
    y.append(x[1]-(len(NTRU)//2 -1)*width - width/2)
else:
    y.append(x[1]-(len(NTRU)//2)*width)

if len(Saber)%2 == 0: # longitud par
    y.append(x[2]-(len(Saber)//2 -1)*width - width/2)
else:
    y.append(x[2]-(len(Saber)//2)*width)
```

```
# Key
plt.figure(1)

fig, ax = plt.subplots()
for i in range(len(Kyber)):
    if i == 1:
        ax.bar(y[0] + i*width, Kyber_key[i], width,
            label='Kyber', facecolor = 'tab:orange',
            edgecolor = 'r')
    else:
        ax.bar(y[0] + i*width, Kyber_key[i], width,
            facecolor = 'tab:orange',edgecolor = 'r')
for i in range(len(NTRU)):
    if i == 1:
        ax.bar(y[1] + i*width, NTRU_key[i], width,
            label='NTRU', facecolor = 'c',
            edgecolor = 'b')
    else:
        ax.bar(y[1] + i*width, NTRU_key[i], width,
            facecolor = 'c',edgecolor = 'b')
for i in range(len(Saber)):
    if i == 1:
        ax.bar(y[2] + i*width, Saber_key[i], width,
            label='Saber',facecolor ='m',
            edgecolor = 'k')
    else:
        ax.bar(y[2] + i*width, Saber_key[i], width,
            facecolor ='m',edgecolor = 'k')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend(loc = 'upper left')
plt.xlabel('Diferent KEMS')
plt.ylabel('Tiempo(nanosegundos)')
plt.title('Key pair')
ax.grid(which='major', color='#EEEEEE', linestyle=':',
        linewidth=0.5)
ax.grid(which='minor', color='#EEEEEE', linestyle=':',
        linewidth=0.5)
ax.minorticks_on()

# Encapsule
plt.figure(2)

fig, ax = plt.subplots()
```

```
for i in range(len(Kyber)):
    if i == 1:
        ax.bar(y[0] + i*width, Kyber_encaps[i], width,
            label='Kyber', facecolor = 'tab:orange',
            edgecolor = 'r')
    else:
        ax.bar(y[0] + i*width, Kyber_encaps[i], width,
            facecolor = 'tab:orange',edgecolor = 'r')
for i in range(len(NTRU)):
    if i == 1:
        ax.bar(y[1] + i*width, NTRU_encaps[i], width,
            label='NTRU', facecolor = 'c',
            edgecolor = 'b')
    else:
        ax.bar(y[1] + i*width, NTRU_encaps[i], width,
            facecolor = 'c',edgecolor = 'b')
for i in range(len(Saber)):
    if i == 1:
        ax.bar(y[2] + i*width, Saber_encaps[i], width,
            label='Saber',facecolor ='m',
            edgecolor = 'k')
    else:
        ax.bar(y[2] + i*width, Saber_encaps[i], width,
            facecolor ='m',edgecolor = 'k')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend(loc = 'upper left')
plt.xlabel('Diferent KEMS')
plt.ylabel('Tiempo(nanosegundos)')
plt.title('Encapsule')
ax.grid(which='major', color='#EEEEEE', linestyle=':',
        linewidth=0.5)
ax.grid(which='minor', color='#EEEEEE', linestyle=':',
        linewidth=0.5)
ax.minorticks_on()

# Decapsule
plt.figure(3)

fig, ax = plt.subplots()
for i in range(len(Kyber)):
    if i == 1:
        ax.bar(y[0] + i*width, Kyber_decaps[i], width,
            label='Kyber', facecolor = 'tab:orange',
```

```
            edgecolor = 'r')
    else:
        ax.bar(y[0] + i*width, Kyber_decaps[i], width,
            facecolor = 'tab:orange',edgecolor = 'r')
for i in range(len(NTRU)):
    if i == 1:
        ax.bar(y[1] + i*width, NTRU_decaps[i], width,
            label='NTRU', facecolor = 'c',
            edgecolor = 'b')
    else:
        ax.bar(y[1] + i*width, NTRU_decaps[i], width,
            facecolor = 'c',edgecolor = 'b')
for i in range(len(Saber)):
    if i == 1:
        ax.bar(y[2] + i*width, Saber_decaps[i], width,
            label='Saber',facecolor ='m',
            edgecolor = 'k')
    else:
        ax.bar(y[2] + i*width, Saber_decaps[i], width,
            facecolor ='m',edgecolor = 'k')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend(loc = 'upper left')
plt.xlabel('Diferent KEMS')
plt.ylabel('Tiempo(nanosegundos)')
plt.title('Decapsule')
ax.grid(which='major', color='#EEEEEE', linestyle=':',
        linewidth=0.5)
ax.grid(which='minor', color='#EEEEEE', linestyle=':',
        linewidth=0.5)
ax.minorticks_on()
```

Bibliografía

[1] M. R. Albrecht, A. Deo, K. G. Paterson. *Cold Boot Attacks on Ring and Module LWE Keys Under the NTT.* Recuperado de https://tches.iacr.org/index.php/TCHES/article/view/7273/6451

[2] A. Banerjee, C. Peikert, A. Rosen. *Pseudorandom Functions and Lattices* (2011). Recuperado de https://web.eecs.umich.edu/ cpeikert/pubs/prf-lattice.pdf

[3] D. J. Bernstein, J. Buchmann, E. Dahmen. *Post-Quantum Cryptography.* Ed. Springer (2009).

[4] H. Carter. Instructor: Chris Peikert. *Lattices in Cryptography.* Georgia Tech, Fall (2013). Recuperado de https://www.studocu.com/en-us/document/georgia-institute-of-technology/lattices-in-cryptography/lecture-notes/lecture-notes-lectures-1-6-everything-you-need-to-know-about-lecture-1-6/744049/view

[5] T. ElGamal. *A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms.* Recuperado de https://link.springer.com/content/pdf/10.1007%2F3-540-39568-7_2.pdf

[6] *European Telecommunications Standards Institute. Quantum Safe Cryptography and Security: An Introduction, Benefits, Enablers and Challenges. White Paper No, 8.* Recuperado de https://portal.etsi.org/Portals/0/TBpages/QSC/Docs/Quantum_Safe_Whitepaper_1_0_0.pdf

[7] M. Hayward. *Quantum Computing and Shor's Algorithm* (2005). Recuperado de http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.121.1509&rep=rep1&type=pdf

[8] J. Hoffstein, J. Pipher, J. H. Silverman. *NTRU: A Ring-Based Public Key Cryptosystem.* Recuperado de https://www.ntru.org/f/hps98.pdf

[9] N. Koblitz. *Elliptic Curve Cryptosystems. Mathematics of Computation*. Volumen 48 (1987). Recuperado de https://www.ams.org/journals/mcom/1987-48-177/S0025-5718-1987-0866109-5/S0025-5718-1987-0866109-5.pdf

[10] M. J. Lucena López. *Criptografía y Seguridad en Computadores*. 3ª Edición. (2003).

[11] V. Lyubashevsky, C. Peikert, O. Regev. *On Ideal Lattices and Learning with Errors Over Rings* (2013). Recuperado de https://eprint.iacr.org/2012/230.pdf

[12] R. J. McEliece. *A Public-Key Cryptosystem Based on Algebraic Coding Theory*. Recuperado de https://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF

[13] R. Michael. *Digitalized Signatures and Public-Key Functions as Intractable as Factorization* (1979). Recuperado de http://publications.csail.mit.edu/lcs/pubs/pdf/MIT-LCS-TR-212.pdf

[14] V. S. Miller. *Use of elliptic curves in cryptography*. Recuperado de https://link.springer.com/content/pdf/10.1007%2F3-540-39799-X_31.pdf

[15] National Institute of Standars and Technology(NIST). *FIPS PUB 202 – SHA-3 standard: Permutation-based hash and extendable-output functions* (2015). Recuperado de https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

[16] NIST. $ReportonPost - QuantumCryptogtaphy$ (2016). Recuperado de http://dx.doi.org/10.6028/NIST.IR.8105

[17] C. Paar, J. Pelzl. *Understanding Cryptography*. Ed. Springer (2010).

[18] J. Patarin. *Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): Two New Families of Asymmetric Algorithms*. Recuperado de https://link.springer.com/content/pdf/10.1007%2F3-540-68339-9_4.pdf

[19] R. Rivest, A. Shamir, and L. Adleman. *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*. Recuperado de http://people.csail.mit.edu/rivest/Rsapaper.pdf

[20] N. P. Smart. *Cryptography Made Simple*. Ed. Springer (2016).

[21] C. Tomás. *Los Teoremas de Minkowki en la geometría de números*. Universidad de Murcia (2014). Recuperado de

https://www.um.es/documents/118351/2874787/TFM_TOMAS+MARTINEZ.pdf/c8354936-21fa-4996-aeab-3ec9794119e7

[22] W. Trappe. L. C. Washington. *Introduction to Cryptography with Coding Theory*. Ed. Pearson. Segunda Edición (2006).

[23] J. Wohlwend. *Elliptic Curve Cryptography: Pre and Post Quantum*. Recuperado de http://math.mit.edu/∼apost/courses/18.204-2016/18.204_Jeremy_Wohlwend_final_paper.pdf

[24] I. Zaballa. *Proyecciones y bases ortonormales.*. UPV-EHU. Recuperado de https://studylib.es/doc/6732452/cap%C2%B4%C4%B1tulo-6-proyecciones-y-bases-ortonormales