Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

Facultad de Ciencias

Doble grado en Matemáticas y Física

CRYSTALS: Kyber y Dilithium, los próximos estándares de criptografía postcuántica

Trabajo Fin de Grado de Matemáticas

Autor:

Olái Prieto Fernández

Tutor:

Ignacio Fernández Rúa

Junio 2023

Índice general

Introducción

Con la publicación del algoritmo de Shor en 1994, así como con los recientes avances en computación cuántica, ha quedado patente que los criptosistemas de clave pública utilizados actualmente tienen fecha de caducidad. Por este motivo, en el año 2016 el NIST (National Institute of Standards and Technology, de EE.UU.) convocó un concurso en el que todas las instituciones académicas e industriales podían participar, con el fin de buscar estándares de criptografía resistentes a los ataques cuánticos, lo que ha recibido el nombre de *criptografía postcuántica*. El concurso tenía dos categorías: firma digital y mecanismo de encapsulamiento de claves (o KEM, por sus siglas en inglés). Actualmente, el concurso se encuentra en la cuarta ronda, con un algoritmo seleccionado para el KEM y 3 finalistas para la firma digital.

El objetivo del presente Trabajo de Fin de Grado es motivar la necesidad de la criptografía postcuántica y explicar los esquemas seleccionados englobados bajo la marca CRYSTALS ("Cryptographic Suite for Algebraic Lattices"), que incluyen el KEM Kyber y el esquema de firma digital Dilithium. Estas técnicas criptográficas han sido desarrolladas conjuntamente por profesionales del Research Institute for Mathematics and Computer Science in the Netherlands, ENS de Lyon, International Business Machines Corporation (IBM), Max Planck Institute for Security and Privacy, NXP Semiconductors, Radboud University y Ruhr Universität Bochum.

El trabajo tiene una alta carga bibliográfica y aborda la construcción de los esquemas Kyber y Dilithium sin requisitos previos, esto es: no se supone conocido ningún concepto relativo a la criptografía. El principal motivo para esto es que no he cursado la asignatura de Códigos Correctores y Criptografía, optativa del cuarto curso del grado, y por tanto yo mismo he debido leer bibliografía y aprender estos conceptos desde cero.

La división por capítulos del trabajo es la siguiente:

En el primer capítulo, se dan definiciones y resultados generales sobre criptografía de clave pública. También se desarrollan las técnicas criptográficas basadas en la dificultad de factorizar números enteros y de resolver el problema del logaritmo discreto, pues son las dos técnicas en las que se basan la mayoría de las comunicaciones a día de hoy. En este capítulo aún no entran en juego ni la computación cuántica ni la criptografía postcuántica. Constituye una introducción a la criptografía para personas que, como yo al empezar el trabajo, no estén familiarizadas con estos conceptos.

El segundo capítulo aborda la computación cuántica. Se da una introducción, también desde cero, a los procedimientos de cómputo de los ordenadores cuánticos. Después se pasa a describir el algoritmo cuántico de Shor que permite factorizar números enteros y resolver el problema del logaritmo discreto, dejando obsoletas las técnicas criptográficas del capítulo 1. Esto pondrá de manifiesto la necesidad de encontrar algoritmos resistentes a ataques cuánticos y desarrollar estándares de criptografía postcuántica.

El tercer capítulo se dedica a la criptografía postcuántica basada en retículos, una de las técnicas más versátiles para el desarrollo de esquemas resistentes a ataques cuánticos. Se expone el concepto de retículo y se dan los criptosistemas y esquemas de firma digital basados en ellos más intuitivos y que primero surgieron históricamente. Posteriormente, se introducen los retículos con estructura algebraica, y se explican sus ventajas y potenciales inconvenientes frente a los retículos sin estructura.

Los capítulos cuarto y quinto describen respectivamente las propuestas Kyber y Dilithium. Son el esquema de KEM seleccionado por el NIST y uno de los tres finalistas para el esquema de firma digital, y basan su funcionamiento en retículos estructurados. Estos capítulos tienen ambos una parte de preliminares, en la que se aclaran algunos conceptos utilizados en su construcción. Posteriormente se detalla su funcionamiento y se dan resultados sobre su corrección y seguridad.

Capítulo 1

Criptografía de clave pública

¿Qué es la criptografía de clave pública?

Una de las principales utilidades de la criptografía es garantizar la privacidad de la comunicación entre dos individuos, habitualmente referidos como Alice y Bob. Imaginemos una situación en que Alice quiere mandarle a Bob un mensaje m. Para ello lo transforma en un c, que llamaremos mensaje cifrado, y lo manda a Bob. A su vez Bob descifra este mensaje cifrado y obtiene de nuevo el mensaje m. Estas transformaciones de cifrado y descifrado que han usado Alice y Bob se determinan de una familia de transformaciones mediante la elección de un parámetro k denominado *clave*. Esta técnica para proteger el secreto de la comunicación se conoce como criptosistema, y formalmente podemos definirla de la siguiente manera:

Definición 1.0.1 (Criptosistema). Un *criptosistema* es una quíntupla $(\mathcal{P},\mathcal{C},\mathcal{K},\mathcal{E},\mathcal{D})$ donde:

- $\mathcal{P}$ es un conjunto finito de posibles mensajes en claro m.
- $\mathcal{C}$ es un conjunto finito de posibles mensajes cifrados c.
- $\mathcal{K}$ es el conjunto finito de posibles claves k.
- Para cada $k \in \mathcal{K}$, hay una función de cifrado $E(\cdot, k) \in \mathcal{E}$ y una función de descifrado $D(\cdot, k) \in \mathcal{D}$ tal que $D(E(m, k), k) = m$ para todo mensaje en claro $m \in \mathcal{P}$.

En algunos criptosistemas, que llamaremos *simétricos* o *de clave privada*, Alice y Bob escogen una clave $k \in \mathcal{K}$, que determina las funciones de cifrado y descifrado. En estos casos $D(\cdot, k)$ se puede derivar directamente de $E(\cdot, k)$. Un criptosistema de este tipo tiene

el inconveniente de que requiere que Alice y Bob se comuniquen previamente por un canal seguro para acordar la clave k que usarán. Si Alice y Bob viven lejos esta primera comunicación segura puede ser difícil de conseguir. Aquí es donde entra en juego la criptografía de clave pública, con los llamados *protocolos de intercambio de claves* y los *criptosistemas de clave pública*.

La idea de un criptosistema de clave pública es que, dada la función de cifrado $E(\cdot, k)$, sea computacionalmente difícil obtener la función de descifrado $D(\cdot, k)$. En este caso, la función de cifrado $E(\cdot, k)$ podría ser publicada en un directorio, de modo que Alice o cualquier persona puedan cifrar mensajes y mandárselos a Bob. Bob sería el único que conocería la función $D(\cdot, k)$, es decir el único capaz de descifrar los mensajes. En esta situación, la función de cifrado sería *pública*, y la de descifrado *privada* o *secreta*. Esta idea fue publicada por primera vez en 1976 por Diffie y Hellman, y desde entonces se han desarrollado multitud de criptosistemas de clave pública, como el RSA o ElGamal.

Este capítulo del TFG se dividirá en tres partes. En primer lugar, se darán las definiciones y resultados generales sobre conceptos como *función resumen*, *función de una vía*, *protocolo de intercambio de claves*, *criptosistema de clave pública* y *esquema de firma digital*. Las dos secciones siguientes estarán dedicadas a ejemplos concretos de estas técnicas criptográficas basadas en la dificultad para factorizar números enteros y en la dificultad para resolver el problema del logaritmo discreto, respectivamente. En general, se sigue la línea de [1], aunque recuperando algunas definiciones y explicaciones de [2].

1.1. Definiciones y resultados generales

1.1.1. Funciones resumen

Las funciones resumen son una primitiva criptográfica; esto es, algoritmos sencillos y bien conocidos sobre los que se construyen técnicas criptográficas más elaboradas. Veamos en primer lugar una definición formal:

Definición 1.1.1 (Función resumen). Una *función resumen* es un algoritmo que toma como entrada un mensaje m y devuelve $t := H(m)$, que recibe el nombre de resumen. Los mensajes se toman en un espacio de mensajes $\mathcal{M}$ y los resúmenes en un espacio de resúmenes $\mathcal{T}$.

Una de las ventajas de las funciones resumen es que los resúmenes $t \in \mathcal{T}$ suelen ser mucho más cortos que los mensajes. Habitualmente, los resúmenes tienen un tamaño prefijado, por

ejemplo 128 ó 256 bits, independientemente de la longitud del mensaje de entrada.

Para una función resumen H, decimos que un par $(m, t) \in \mathcal{M} \times \mathcal{T}$ es un par válido si $t = H(m)$. Es deseable que la única forma de producir un par válido sea elegir $m \in \mathcal{M}$ y aplicarle H, pero en la práctica usaremos una noción algo más débil, que es que el *problema de colisión*, definido a continuación, sea difícil de resolver:

Definición 1.1.2 (Problema de colisión). Dada una función resumen $H(\cdot)$, el problema de colisión consiste en encontrar $m, m' \in \mathcal{M}$ tales que $m' \neq m$ y $H(m) = H(m')$.

Notemos que, como típicamente $\mathcal{T}$ es más pequeño que $\mathcal{M}$, las colisiones ocurren, pero exigiremos que sean difíciles de encontrar.

Un grupo de funciones resumen célebres son las que conforman el estándar del NIST, englobadas bajo la marca SHA (Secure Hash Algoritm), y que tienen como entrada y salida cadenas de bits. Además, la versión SHA3 utiliza la llamada construcción de *esponja*, lo que permite que las cadenas de bits que actúan como entrada y salida tengan longitud variable. Algunas funciones de esta versión, como SHA3-256 admiten entradas de longitud variable y dan siempre salidas de un número fijo de bits, 256 en este caso. Otras funciones, que pertenecen a la familia SHAKE, pueden dar salidas de longitud variable. Para todas las funciones de la versión SHA3, a día de hoy se considera prácticamente imposible encontrar colisiones.

El modelo del oráculo aleatorio

En esta sección describiré una función resumen 'ideal'; esto es, un modelo teórico al que deberían aproximarse las funciones resumen usadas en la práctica. Esta función recibe el nombre de 'oráculo aleatorio'.

La idea es elegir aleatoriamente una función $\mathcal{O}$ con dominio $\mathcal{M}$ y codominio $\mathcal{T}$, y acceder a ella únicamente mediante un 'oráculo' o 'caja negra'. Esto quiere decir que no se nos da una fórmula o un algoritmo para obtener los valores de $\mathcal{O}$. Únicamente podemos preguntarle al oráculo, que nos dará un valor de $\mathcal{O}(m)$ para el mensaje m que le pasemos. Dicho valor debería ser aleatorio, salvo que ya le hayamos pasado antes el mismo mensaje m, en cuyo caso debe darnos el mismo valor que antes. Una función de estas características es demasiado costosa de implementar, por lo que en la práctica nuestras funciones resumen nunca serán auténticos oráculos aleatorios. Se trata en cualquier caso de un modelo teórico muy útil

para definir la seguridad de cualquier técnica criptográfica que utilice una función resumen. Esto se hace de la siguiente manera:

Supongamos que tenemos una técnica criptográfica $\mathcal{S}$, en cuya implementación utilizamos una función resumen $H : \mathcal{M} \longrightarrow \mathcal{T}$. En criptografía, cuando se quiere analizar la seguridad teórica de una propiedad X sobre $\mathcal{S}$, planteamos un 'juego' entre un 'retador' y un 'adversario' arbitrario $\mathcal{A}$. Este juego será una abstracción teórica para el ataque a la técnica criptográfica, y permite dejar claro la capacidad del adversario y sus objetivos con el ataque. La figura de 'adversario' modeliza al posible atacante, mientras que el 'retador' suplanta a todos los participantes en la técnica. Por ejemplo, en un criptosistema el retador hará el papel tanto de Alice como de Bob. La idea es que el adversario le hace unas preguntas al retador, que para responderlas deberá calcular funciones relacionadas con la técnica $\mathcal{S}$, que pueden involucrar a la función resumen H. El juego define una ventaja $\mathrm{Xadv}[\mathcal{A}, \mathcal{S}]$, que representa la probabilidad de que el adversario cumpla su objetivo, y la seguridad de la propiedad X en la técnica $\mathcal{S}$ significará que $\mathrm{Xadv}[\mathcal{A}, \mathcal{S}]$ sea despreciable (por ejemplo, menor que 2^{-100}) para cualquier adversario.

La técnica $\mathcal{S}$ evalúa H en los puntos que quiera, sin importarle la implementación interna de H. Podemos decir por tanto que $\mathcal{S}$ usa H como un oráculo, y tiene sentido evaluar la seguridad de la propiedad X 'en el modelo del oráculo aleatorio'. En este caso, el juego de ataque descrito anteriormente se modificará ligeramente, de modo que la función resumen 'real' H sea sustituida por un oráculo aleatorio $\mathcal{O}$. Tanto retador como adversario solo tendrán acceso a esta función mediante el oráculo. En concreto, el juego se modifica de la siguiente manera:

- Al inicio del juego, el retador elige aleatoriamente $\mathcal{O} : \mathcal{M} \longrightarrow \mathcal{T}$.
- Además de las preguntas permitidas en el juego anterior, el adversario $\mathcal{A}$ puede realizar preguntas de oráculo aleatorio; es decir, le pasará un mensaje $m \in \mathcal{M}$ al retador, que le devuelve $t = \mathcal{O}(m)$. El adversario puede hacer tantas preguntas de oráculo aleatorio como quiera.
- En las preguntas del juego anterior, el retador da la respuestas usando la función $\mathcal{O}$ en lugar de H.

La ventaja del adversario se define como antes, y la denotaremos $\mathrm{X}^{\mathrm{ro}}\mathrm{Adv}[\mathcal{A}, \mathcal{S}]$. La seguridad de la propiedad X *en el modelo del oráculo aleatorio* significará que $\mathrm{X}^{\mathrm{ro}}\mathrm{Adv}[\mathcal{A}, \mathcal{S}]$

sea despreciable para cualquier adversario. La idea de este enfoque es estudiar la seguridad de $\mathcal{S}$ sin supeditarla a la implementación concreta de H, sino usando un modelo teórico ideal.

1.1.2. Intercambio de claves

Volvamos al problema planteado al inicio del capítulo. Alice y Bob quieren intercambiar mensajes y quieren hacerlo de forma confidencial, pero nunca se han visto en persona y por tanto no han podido acordar una clave. Lo primero que pueden hacer es utilizar algún método para acordar una clave, procurando que esta clave no sea conocida por un posible adversario.

Para formalizar esta idea, modelizamos a Alice y a Bob como dos máquinas probabilísticas (esto es, capaces de desarrollar algoritmos que hacen uso del azar) A y B que se comunican. Un *protocolo de intercambio de claves* P es un par de máquinas (A, B) que se mandan mensajes de forma alternada. Diremos que el protocolo es anónimo, pues en realidad el protocolo en sí mismo no asegura que A esté realmente comunicándose con B y viceversa. La idea es que al acabar el protocolo, ambas máquinas posean el mismo valor k. La *transcripción del protocolo* T_P es la secuencia de mensajes intercambiados por las máquinas durante el mismo. Definimos la seguridad del protocolo de intercambio de claves utilizando el siguiente juego:

Juego 1.1.1 (Intercambio de claves anónimo). Para un protocolo de intercambio de claves $P = (A, B)$ y dado un adversario $\mathcal{A}$, el juego de desarrolla como sigue:

- El retador desarrolla el protocolo entre A y B para generar una clave k y una transcripción T_P. Le da T_P al adversario $\mathcal{A}$.
- El adversario $\mathcal{A}$ elige $\hat{k}$, la clave que cree que es k.

Definimos la ventaja de $\mathcal{A}$, denotada $\mathrm{AnonKEavd}[\mathcal{A}, P]$, como la probabilidad de que $\hat{k} = k$, y diremos que el protocolo anónimo de intercambio de claves es 'seguro' si para todos los adversarios eficientes $\mathcal{A}$, la cantidad $\mathrm{AnonKEavd}[\mathcal{A}, P]$ es despreciable.

Observación. Por adversario 'eficiente' entenderemos un adversario que ejecute algoritmos con tiempos de ejecución polinomiales en el tamaño de la entrada.

¿Cómo se pueden desarrollar protocolos de intercambio de claves seguros? Una herramienta muy útil son las llamadas funciones de una vía con trampa, que se tratan a continuación. Esta herramienta no es la única: existen protocolos de intercambio de claves que no basan en esto su funcionamiento, como por ejemplo el protocolo de intercambio de claves de Diffie-Hellman, uno de los más importantes, y que veremos más adelante en este capítulo.

Funciones de una vía con trampa

A grandes rasgos, una función de una vía es una función que es fácil de computar pero difícil de invertir. En la actualidad, hay muchas funciones que se cree que son de una vía, aunque no hay muchas que esté probado que efectivamente lo sean. En el contexto de la criptografía de clave pública, es conveniente además que la función de una vía tenga una trampa; esto es, una información que permita invertirla fácilmente, mientras que sin esta información siga siendo muy difícil invertirla. Formalmente, definimos primero lo que es una función con trampa:

Definición 1.1.3 (Esquema de función con trampa). Sean $\mathcal{X}$ y $\mathcal{Y}$ conjuntos finitos. Un *esquema de función con trampa* $\mathcal{T}$ es una tupla de algoritmos (G, F, I) donde:

- G es un algoritmo probabilistico de generación de claves, que es invocado como $(pk, sk) \leftarrow G(\cdot)$, donde pk es la *clave pública* y sk la *clave privada*. Así, cada ejecución de $G(\cdot)$ devuelve un par clave pública - clave privada.

- F es un algoritmo determinístico que es invocado como $y = F(pk, x) \in \mathcal{Y}$, donde pk es una clave pública y $x \in \mathcal{X}$. De este modo, para cada pk fija, $F(\cdot, pk)$ es una función de $\mathcal{X}$ en $\mathcal{Y}$.

- I es un algoritmo determinístico que es invocado como $x = I(sk, y) \in \mathcal{X}$, donde sk es una clave privada e $y \in \mathcal{Y}$; es decir, para cada sk fija, $I(\cdot, sk)$ es una función de $\mathcal{Y}$ en $\mathcal{X}$.

Además, se debe satisfacer la siguiente propiedad de corrección: para todos las posibles salidas (pk, sk) de $G(\cdot)$, y para todo $x \in \mathcal{X}$, se tiene que $I(sk, F(pk, x)) = x$.

Observación. La propiedad de corrección recoge la idea de que sk es una trampa para invertir $F(pk, \cdot)$.

Observación. En el caso en que $\mathcal{X}$=$\mathcal{Y}$, $F(pk, \cdot)$ es una biyección de $\mathcal{X}$ en sí mismo. En este caso, diremos que (G, F, I) es un 'esquema de permutación con trampa' sobre $\mathcal{X}$.

Formalicemos ahora el concepto de función de una vía con trampa. Para ello se define el siguiente juego entre retador y adversario:

Juego 1.1.2 (Función de una vía con trampa). Dado un esquema de función con trampa $\mathcal{T} = (G, F, I)$, definido sobre $(\mathcal{X}, \mathcal{Y})$ y un adversario $\mathcal{A}$, se desarrolla el siguiente juego:

- El retador computa $(pk, sk) \leftarrow G(\cdot)$. Después, elige aleatoriamente un $x \in \mathcal{X}$, y calcula $y = F(pk, x)$. Le manda (pk, y) al adversario.
- El adversario elige un $\hat{x} \in \mathcal{X}$ que cree que verifica $I(sk, y) = \hat{x}$.

Definimos la ventaja del adversario en invertir $\mathcal{T}$, y la denotamos $\mathrm{OWadv}[\mathcal{A}, \mathcal{T}]$, como la probabilidad de que $x = \hat{x}$.

Diremos que un esquema de función con trampa $\mathcal{T}$ es de una vía si para todos los adversarios eficientes $\mathcal{A}$, la cantidad $\mathrm{OWadv}[\mathcal{A}, \mathcal{T}]$ es despreciable.

Intercambio de claves a partir de una función de una vía con trampa

Como ya se ha mencionado, un esquema de función de una vía con trampa $\mathcal{T} = (G, F, I)$, definido sobre $(\mathcal{X}, \mathcal{Y})$, se puede utilizar, entre otras cosas, para construir un protocolo anónimo de intercambio de claves que sea seguro. Se procede de la siguiente manera:

- Alice computa $(pk, sk) \leftarrow G(\cdot)$ y le manda pk a Bob.
- Cuando recibe pk de Alice, Bob elige al azar uniformemente un $x \in \mathcal{X}$, calcula $y = F(pk, x)$ y se lo manda a Alice.
- Cuando recibe y de Bob, Alice calcula $x = I(sk, y)$.

La propiedad de corrección garantiza que al final del protocolo Alice y Bob están en posesión del mismo elemento x de $\mathcal{X}$. En relación a la seguridad, notar que un adversario que pudiera obtener x estaría en condiciones de ganar el juego 1.1.2. Por tanto, la seguridad del esquema de función de una vía con trampa garantiza la seguridad del protocolo de intercambio de claves.

1.1.3. Criptosistemas de clave pública

Como hemos visto, Alice y Bob pueden desarrollar un protocolo de intercambio de claves para acordar una clave con la que enviarse mensajes cifrados de clave privada. En esta sección veremos que Alice puede mandarle mensajes cifrados a Bob aunque no compartan una clave privada, utilizando para ello un *criptosistema* o *esquema de cifrado de clave pública*. Definiremos formalmente este concepto, y trataremos las nociones de seguridad que es deseable que tengan.

Definición 1.1.4 (Criptosistema de clave pública). Sean $\mathcal{M}$ y $\mathcal{C}$ conjuntos finitos, que usaremos como espacio de mensajes en claro y espacio de mensajes cifrados, respectivamente. Un *criptosistema de clave pública* $\varepsilon = (G, E, D)$ definido sobre $(\mathcal{M}, \mathcal{C})$ es una tupla de algoritmos eficientes:

- G, el algoritmo de generación de claves, es un algoritmo probabilístico invocado como $(pk, sk) \leftarrow G(\cdot)$, donde pk será la clave pública y sk la clave privada.
- E, el algoritmo de cifrado, es un algoritmo probabilístico invocado como $c \leftarrow E(pk, m)$, donde pk es la clave pública, $m \in \mathcal{M}$ y $c \in \mathcal{C}$.
- D, el algoritmo de descifrado, es un algoritmo determinístico invocado como $m = D(sk, c)$, donde sk es la clave privada, c es un mensaje cifrado de $\mathcal{C}$ y m es, o bien un mensaje en claro de $\mathcal{M}$, o bien un carácter especial de rechazo (que debe ser diferente a todos los elementos de $\mathcal{M}$, generalmente se toma el carácter $\perp$), que simula un 'error de descifrado'.

Requerimos la siguiente propiedad de corrección: para todas las posibles salidas (pk, sk) de $G(\cdot)$, y para todo $m \in \mathcal{M}$, debemos tener $Pr[D(sk, E(pk, m)) = m] = 1$.

Una propiedad algo más débil que esta, y que basta en algunos criptosistemas, es la noción de ser $(1 - \delta)$-correcto. Decimos que el criptosistema es $(1 - \delta)$-correcto si

$$\mathbf{E}\left[\max_{m \in \mathcal{M}} P(D(sk, E(pk, m)) = m)\right] \geq 1 - \delta$$

donde la esperanza se toma sobre $(pk, sk) \leftarrow G(\cdot)$ y la probabilidad de dentro sobre la aleatoridad de E. La idea es que, de media, para cada pareja clave pública - clave privada, siempre existe algún mensaje que será descifrado correctamente con una probabilidad superior a $1 - \delta$.

Algunas nociones de seguridad y sus consecuencias más inmediatas

Seguridad semántica

Se introduce a continuación la noción de seguridad semántica para esquemas de cifrado de clave pública. Intuitivamente, esta noción quiere decir que ningún adversario es capaz de distinguir el cifrado de dos mensajes en claro de su elección o, dicho de otro modo, que el texto cifrado no proporciona información sobre el texto en claro.

Juego 1.1.3 (Seguridad semántica). Dado un criptosistema de clave pública $\varepsilon = (G, E, D)$ definido sobre $(\mathcal{M}, \mathcal{C})$, y dado un adversario $\mathcal{A}$, definimos dos experimentos. Para $b = 0, 1$, definimos el experimento b como:

- El retador calcula $(pk, sk) \leftarrow G(\cdot)$ y le manda pk al adversario.
- El adversario elige mensajes $m_0, m_1 \in \mathcal{M}$ de la misma longitud, y se los manda al retador.
- El retador calcula $c \leftarrow E(pk, m_b)$, y le manda c al adversario.
- El adversario elige un bit $\hat{b} \in 0, 1$, que cree que es b.

Si W_b es el suceso en que $\mathcal{A}$ saca 1 en el experimento b, definimos la ventaja de $\mathcal{A}$ con respecto a ε como $\mathrm{SSavd}[\mathcal{A}, \varepsilon] := |Pr[W_o] - Pr[W_1]|$. Esta probabilidad está definida en el espacio de probabilidad definido por la aleatoriedad de los algoritmos G y E, así como por las elecciones de mensajes hechas por el adversario.

Definición 1.1.5. Diremos que un criptosistema de clave pública ε es *semánticamente seguro* si para todos los adversarios eficientes $\mathcal{A}$, la cantidad $\mathrm{SSavd}[\mathcal{A}, \varepsilon]$ es despreciable.

Consecuencia: *El algoritmo de cifrado E debe ser probabilístico.* ¿Por qué en la definición de criptosistema de clave pública hemos exigido que E sea probabilístico? Porque en caso contrario ε no sería semánticamente seguro. En efecto, si $|\mathcal{M}| \geq 2$ y E es determinístico, el siguiente adversario $\mathcal{A}$ rompería la seguridad semántica de $\varepsilon = (G, E, D)$:

- $\mathcal{A}$ recibe una clave pública del retador.
- $\mathcal{A}$ elige dos mensajes distintos $m_0, m_1 \in \mathcal{M}$ y se los manda al retador. El retador responde con $c := E(pk, m_b)$ para algún $b \in 0, 1$.
- $\mathcal{A}$ calcula $c_0 := E(pk, m_0)$ y saca 0 si $c = c_0$. En otro caso saca 1.

Como E es determinístico, sabemos que siempre que $b = 0$ tendremos $c = c_0$. Por tanto, siempre que $b = 0$ el adversario sacará un 0, y siempre que $b = 1$ el adversario sacará un 1. Así pues, $\text{SSavd}[\mathcal{A}, \varepsilon] = 1$, y el criptosistema no es semánticamente seguro.

Debemos notar que esto solo es necesario en criptosistemas de clave pública. Existen criptosistemas de clave privada, tratados muy por encima en este TFG, que son semánticamente seguros utilizando algoritmos de cifrado determinísticos.

CPA seguridad

Se introduce a continuación la noción de seguridad semántica frente a ataques de mensaje en claro escogido, más conocida como CPA seguridad (del inglés: chosen plaintext attack). Una vez más, recurrimos a un juego entre adversario y retador para la formalización de este concepto:

Juego 1.1.4 (CPA seguridad). Dado un criptosistema de clave pública $\varepsilon = (G, E, D)$ definido sobre $(\mathcal{M}, \mathcal{C})$, y dado un adversario $\mathcal{A}$, definimos dos experimentos. Para $b = 0, 1$, definimos el experimento b como:

- El retador calcula $(pk, sk) \leftarrow G(\cdot)$ y manda pk al adversario.
- El adversario envía una serie de preguntas al retador: Para $i = 1, 2, \ldots,$ la pregunta $i-$ésima consiste en un par de mensajes $m_{i0}, m_{i1} \in \mathcal{M}$ de la misma longitud. El retador calcula $c_i \leftarrow E(pk, m_{ib})$ y manda c_i al adversario.
- El adversario elige un bit $\hat{b} \in 0, 1$, que cree que es b.

Si W_b es el suceso en que el adversario $\mathcal{A}$ saca un 1 en el experimento b, entonces se define la ventaja de $\mathcal{A}$ con respecto a ε como $\text{CPAavd}[\mathcal{A}, \varepsilon] := |Pr[W_0] - Pr[W_1]|$.

Observación. Existen otras formas de denotar y de definir este ventaja. Por ejemplo, en [13] se denota $\text{Adv}_{\varepsilon}^{cpa}$ y se define como $\text{Adv}_{\varepsilon}^{cpa}(\mathcal{A}) = |Pr[b = \hat{b}] - 1/2|$.

Definición 1.1.6. Un criptosistema de clave pública ε es *CPA seguro* si para todo adversario eficiente $\mathcal{A}$, la cantidad $\text{CPAadv}[\mathcal{A}, \varepsilon]$ (ó $\text{Adv}_{\varepsilon}^{cpa}(\mathcal{A})$) es despreciable. Se puede probar que la noción de CPA seguridad coincide para ambas definiciones de la ventaja.

Un teorema que no se prueba en este TFG establece que si un criptosistema de clave pública ε es semánticamente seguro, entonces también es CPA seguro. Una prueba de este resultado se puede encontrar en [1]. Intuitivamente, este propiedad se debe a que en un

criptosistema de clave pública, un adversario puede cifrar tantos mensajes como quiera, y no necesita por tanto hacerle preguntas de cifrado al retador.

CCA seguridad

Existe también una noción de seguridad frente a ataques de mensaje cifrado escogido, más conocida como CCA seguridad (del inglés: chosen ciphertext attack). Que un criptosistema sea CCA seguro significa que un adversario es incapaz de distinguir entre los mensajes cifrados de dos textos en claro, aun teniendo acceso al algoritmo de descifrado. Se define a continuación como un juego entre adversario y retador:

Juego 1.1.5 (CCA seguridad). Dado un criptosistema de clave pública $\varepsilon = (G, E, D)$ definido sobre $(\mathcal{M}, \mathcal{C})$, y dado un adversario $\mathcal{A}$, definimos dos experimentos. Para $b = 0, 1$, definimos el experimento b como:

- El retador calcula $(pk, sk) \leftarrow G(\cdot)$ y envía pk al adversario.
- El adversario $\mathcal{A}$ hace una serie de preguntas al retador. Cada pregunta puede ser de dos tipos:
  - *Preguntas de cifrado*: para $i = 1, 2, \ldots$, la $i-$ésima pregunta de cifrado consiste en un par de mensajes $m_{i0}, m_{i1} \in \mathcal{M}$ de la misma longitud. El retador calcula $c_i \leftarrow E(pk, m_{ib})$ y se lo manda al adversario.
  - *Preguntas de descifrado*: para $j = 1, 2, \ldots$, la $j-$ésima pregunta de descifrado consiste en un texto cifrado $\hat{c}_j \in \mathcal{C}$ que no esté entre las respuestas a preguntas de cifrado previas. El retador calcula $\hat{m}_j = D(sk, \hat{c}_j)$ y se lo manda al adversario.
- Al final del juego, el adversario elige un bit $\hat{b} \in 0, 1$, que cree que es b.

Si W_b es el suceso en que el adversario $\mathcal{A}$ saca un 1 en el experimento b, definimos la ventaja de $\mathcal{A}$ con respecto a ε como $\text{CCAadv}[\mathcal{A}, \varepsilon] := |Pr[W_0] - Pr[W_1]|$.

Observación. Como para la CPA seguridad, existen otras formas de denotar y de definir este ventaja. De nuevo, en [13] se denota $\text{Adv}^{cca}_{\varepsilon}$ y se define como $\text{Adv}^{cca}_{\varepsilon}(\mathcal{A}) = |Pr[b = \hat{b}] - 1/2|$.

Definición 1.1.7 (CCA seguridad). Un criptosistema de clave pública ε es *CCA seguro* si para todos los adversario eficientes $\mathcal{A}$, la cantidad $\text{CCAadv}[\mathcal{A}, \varepsilon]$ (ó $\text{Adv}^{cca}_{\varepsilon}(\mathcal{A})$) es despreciable. Una vez más, la noción de CCA seguridad es la misma para las dos definiciones de ventaja.

En general, lo más deseable es desarrollar criptosistemas CCA seguros, ya que recoge la idea de que el cifrado es adecuado aún cuando el adversario tiene acceso a la 'máquina de descifrar', lo que le otorga un gran poder. Conseguir criptosistemas CCA seguros puede resultar complicado, y en ocasiones se construyen criptosistemas CPA seguros. Después, se utilizan transformaciones generales que crean criptosistemas CCA seguros a partir de otros CPA seguros, como la transformación de Fujisaki-Okamoto [3].

1.1.4. Esquemas de firma digital

Veamos ahora lo que es y cómo funciona una firma digital. A grandes rasgos, una firma digital es un modo de firmar documentos de manera electrónica; es decir, de especificar la persona que es responsable del documento. ¿Cómo funciona y qué relación guarda con una firma convencional, escrita a mano? Existe en primer lugar una diferencia: mientras que una firma convencional es parte del documento firmado, una firma electrónica va aparte, por lo que deberemos vincularla de algún modo al documento. El proceso de firma y verificación de la firma, en cambio, guarda bastantes similitudes: una firma convencional solo puede ser hecha por una persona, que se supone que es la única capaz de reproducir siempre el mismo trazo, pero puede ser verificada por cualquiera, por ejemplo comparando con la firma que aparece en el DNI. Las firmas digitales seguirán el mismo esquema: sólo una persona podrá producirlas, pero cualquiera podrá verificarlas. A continuación vemos una definición formal.

Definición 1.1.8 (Esquema de firma digital). Un *esquema de firma digital* $\mathcal{S} = (G, S, V)$ es un tupla de algoritmos eficientes tales que:

- G, el algoritmo de generación de claves, es un algoritmo probabilístico invocado como $(pk, sk) \leftarrow G(\cdot)$, que crea sk, una clave secreta para firmar, y pk, una clave pública para verificar.

- S, el algoritmo de firma, es un algoritmo probabilístico invocado como $\sigma \leftarrow S(sk, m)$, donde m es un mensaje. σ es lo que se conoce como la firma.

- V, el algoritmo de verificación, es un algoritmo determinístico invocado como $V(pk, m, \sigma)$. Debe devolver `accept` o `reject`.

Requeriremos que una firma generada por S sea siempre aceptada por V. Esto es, para todo

par (pk, sk) producido por G y todos los mensajes m, tendremos

$$Pr[V(pk, m, S(sk, m)) = \texttt{accept}] = 1.$$

Además, diremos que los mensajes están en un conjunto de mensajes finito $\mathcal{M}$, y que las firmas están en un espacio de firmas finito Σ. En este caso, diremos que $\mathcal{S} = (G, S, V)$ está definido sobre $(\mathcal{M}, \Sigma)$.

La idea es que sólo la persona en posesión de la clave secreta sk podrá ejecutar el algoritmo de firma, pero cualquiera con la clave pública pk podrá ejecutar el de verificación.

Seguridad para esquemas de firma digital

A la hora de hablar de la seguridad de un esquema de firma digital, se suelen considerar tres tipos de ataques:

- *Key-only attacks (ataques de sólo clave)*: el adversario únicamente conoce la clave pública pk.
- *Known message attack (ataques de mensaje conocido)*: el adversario conoce una serie de pares mensaje-firma $(m_1, \sigma_1), (m_2, \sigma_2), \ldots$.
- *Chosen message attack (ataque de mensaje escogido)*: el adversario elige unos mensajes $m_1, m_2, \ldots$ y pide su firma al retador.

A su vez, un ataque puede perseguir distintos objetivos:

- *Total break (roto total)*: El adversario es capaz de determinar la clave privada.
- *Selective forgery (falsificación selectiva):* Con una probabilidad no nula, el adversario es capaz de crear un par mensaje-firma válido para un mensaje de su elección que no haya sido firmado anteriormente por el retador.
- *Existential forgery (falsificación existencial):* El adversario es capaz de crear un par mensaje-firma válido para un mensaje cualquiera que no haya sido firmado anteriormente. Este mensaje puede ser una cadena de caracteres aleatoria y sin sentido.

Idealmente, la noción de seguridad que exigiremos es que el adversario sea incapaz de crear una falsificación existencial con ataques de mensaje escogido, que es la que se corresponde con un mayor poder del adversario. Formalicémoslo con un juego:

Juego 1.1.6. Dado un esquema de firma digital $\mathcal{S} = (G, S, V)$ definido sobre $(\mathcal{M}, \Sigma)$ y un adversario $\mathcal{A}$, el juego se desarrolla de la siguiente manera.

- El retador ejecuta el algoritmo G y obtiene pk y sk. Le manda pk a $\mathcal{A}$.
- $\mathcal{A}$ hace varias preguntas al retador. Para $i = 1, 2, \ldots$, la pregunta $i-$ésima es un mensaje $m_i \in \mathcal{M}$. El retador calcula $\sigma_i \leftarrow S(sk, m_i)$ y luego le da σ_i a $\mathcal{A}$. En este paso es en el que se hace el ataque con mensajes escogidos.
- En su caso, el adversario $\mathcal{A}$ presenta un candidato a falsificación $(m, \sigma) \in \mathcal{M} \times \Sigma$.

Diremos que el adversario gana el juego si $V(pk, m, \sigma) = \texttt{accept}$, y m es nuevo, esto es $m \notin \{m_1, m_2, \ldots\}$. Esto significa que el par (m, σ) es una *falsificación existencial*.

Definimos la ventaja de $\mathcal{A}$ con respecto a $\mathcal{S}$, denotada $\mathrm{SIGadv}[\mathcal{A}, \mathcal{S}]$ ó $\mathrm{Adv}_{\mathcal{S}}^{UF-CMA}(\mathcal{A})$, como la probabilidad de que $\mathcal{A}$ gane el juego.

Definición 1.1.9 (Esquema de firma digital seguro). Diremos que un esquema de firma digital $\mathcal{S}$ es *existencialmente infalsificable bajo ataques de mensaje escogido* (UF-CMA) o, simplemente, *seguro* si para todo adversario eficiente $\mathcal{A}$ la cantidad $\mathrm{SIGadv}[\mathcal{A}, \mathcal{S}]$ es despreciable.

Esta definición asegura que si un esquema de firma digital es seguro, entonces es difícil generar pares mensaje-firma válidos. Notemos que la noción de UF-CMA seguridad considera que un esquema de firma digital sigue siendo *seguro* incluso si el adversario puede transformar un par válido (m, σ) en otro par válido (m', σ). Una noción de seguridad más fuerte que también exige que estas falsificaciones no se produzcan es la noción de ser *fuertemente existencialmente infalsificable bajo ataques de mensaje escogido* (SUF-CMA), que viene formalizada en el siguiente juego:

Juego 1.1.7 (SUF-CMA seguridad). Dado un esquema de firma digital $\mathcal{S} = (G, S, V)$ definido sobre $(\mathcal{M}, \Sigma)$ y un adversario $\mathcal{A}$, el juego es idéntico al anterior. La diferencia está en que diremos que el adversario gana el juego si $V(pk, m, \sigma) = \texttt{accept}$ y (m, σ) es nuevo, esto es $(m, \sigma) \notin \{(m_1, \sigma_1), (m_2, \sigma_2), \ldots\}$.

Definimos la ventaja de $\mathcal{A}$ con respecto a $\mathcal{S}$, denotada $\mathrm{stSIGadv}[\mathcal{A}, \mathcal{S}]$ ó $\mathrm{Adv}_{\mathcal{S}}^{SUF-CMA}(\mathcal{A})$, como la probabilidad de que $\mathcal{A}$ gane el juego.

Incluso con la noción de ser SUF-CMA seguro, hay algunos aspectos que desearíamos de una firma que se escapan. En primer lugar, no exige una correspondencia unívoca entre

un mensaje y su firma. Así, dos mensajes $m \neq m'$ pueden tener la misma firma σ. Esto plantea un problema: la persona firmante podría asegurar que ella no generó el par válido (m, σ), sino el par (m', σ), y no habría forma de saber si dice la verdad. Por este motivo, muchos esquemas de firma digital sí relacionan mensaje y firma.

Por otra parte, un esquema de firma digital SUF-CMA seguro puede ser vulnerable a DSKS (Duplicate Signature Key Selection). Esto significa que dado un par (m, σ) válido bajo la clave pública pk, un atacante podría general un par de claves pk' y sk' tales que (m, σ) sea válido bajo la clave pk'. Esto se puede solucionar si el firmante adjunta su clave pública al mensaje antes de firmarlo. De esta manera, la clave pública es verificada al mismo tiempo que el mensaje. En general, añadir la clave pública al mensaje firmado es una buena práctica y se recomienda en el mundo real.

El paradigma hash-and-sign

En la práctica, debemos poder firmar mensajes de longitud arbitraria. No obstante, un esquema de firma digital seguro puede estar definido sobre un espacio más pequeño, como por ejemplo $\{0,1\}^{256}$. Utilizando una función resumen, conseguiremos firmar mensajes de longitud mucho mayor. Se procede como sigue:

Sea $\mathcal{S} = (G, S, V)$ un esquema de firma digital definido sobre $(\mathcal{M}, \Sigma)$, y sea $H : \mathcal{M}' \longrightarrow \mathcal{M}$ una función resumen. Entonces se puede definir un nuevo esquema de firma digital $\mathcal{S}' = (G', S', V')$ sobre $(\mathcal{M}', \Sigma)$ dado por

$$S'(sk, m) := S(sk, H(m)) \qquad\qquad V'(pk, m, \sigma) := V(pk, H(m), \sigma).$$

El siguiente teorema establece bajo qué condiciones el esquema de firma digital $\mathcal{S}'$ es seguro:

Teorema 1.1.1. Sea $\mathcal{S}$ un esquema de firma digital seguro definido sobre $(\mathcal{M}, \Sigma)$ y sea $H : \mathcal{M}' \longrightarrow \mathcal{M}$ una función resumen resistente a colisiones. Entonces el esquema de firma digital $\mathcal{S}'$ sobre $(\mathcal{M}', \Sigma)$ definido anteriormente es seguro.

Haciendo pues uso de funciones resumen conocidas como SHA3-256, basta centrarse en conseguir esquemas de firma digital seguros para mensajes de 256 bits.

1.2. Criptografía basada en la dificultad de factorizar números enteros

Poco después de que Diffie y Hellman publicasen su idea de la criptografía de clave pública, Rivest, Shamir y Adleman idearon un criptosistema basado en la dificultad de factorizar números enteros. Por las iniciales de sus creadores, recibió el nombre de cripstosistema RSA, y hoy en día es una de las técnicas más habituales dentro de la criptografía de clave pública. En esta sección del TFG, comenzaremos viendo cómo usar esta dificultad para generar funciones de una vía con trampa, para después aplicarlo a intercambio de claves, a la generación de un criptosistema de clave pública CCA seguro y a un esquema de firma digital. Usaremos resultados y técnicas algebraicas conocidas, como el algoritmo euclídeo extendido o el pequeño teorema de Fermat.

1.2.1. Un esquema de permutación con trampa. La suposición RSA

Definamos en primer lugar el algoritmo probabilístico RSAGen, que toma como entradas un entero $l > 2$ y un entero impar $e > 2$, y que devuelve claves 'de tipo RSA'.

Algoritmo 1.2.1 ($RSAGen$). $RSAGen(l, e) :=$

- Generar un primo aleatorio p de l bits tal que $mcd(e, p-1) = 1$.
- Generar un primo aleatorio q de l bits tal que $mcd(e, q-1) = 1$ y $q \neq p$.
- $n = pq$.
- $d \equiv e^{-1} \bmod (p-1)(q-1)$.
- Devolver (n, d).

Para calcular $d \equiv e^{-1} \bmod (p-1)(q-1)$ usaremos el algoritmo euclídeo extendido. Notemos que este inverso existe pues hemos impuesto $mcd(e, p-1) = 1$ y $mcd(e, q-1) = 1$, y por tanto $mcd(e, (p-1)(q-1)) = 1$. Usaremos este algoritmo para definir el esquema de permutación con trampa $\mathcal{T}_{RSA}$:

Definición 1.2.1. Dados unos valores de l y e como en el algoritmo anterior, se define el *esquema de permutación con trampa RSA* como $\mathcal{T}_{RSA} = (G, F, I)$ donde:

- El algoritmo G funciona de la siguiente manera:

$$(n, d) \leftarrow RSAGen(l, e), \quad pk = (n, e), \quad sk = (n, d). \text{ Devolver } (pk, sk).$$

- Dada una clave pública $pk = (n, e)$ y $x \in \mathbb{Z}_n$, se define $F(pk, x) := x^e \in \mathbb{Z}_n$.

- Dada una clave secreta $sk = (n, d)$ e $y \in \mathbb{Z}_n$, se define $I(sk, y) := y^d \in \mathbb{Z}_n$.

Observación. Para cada $pk = (n, e)$, la función $F(pk, \cdot)$ es una función de $\mathbb{Z}_n$ en $\mathbb{Z}_n$. Esto quiere decir que para cada pk el dominio y el rango de $F(pk, \cdot)$ cambia, lo que no estaba contemplado en nuestra definición de esquema de permutación con trampa. Veremos que esta inexactitud técnica no nos impedirá utilizar RSA con éxito.

¿Satisface este esquema $\mathcal{T}_{RSA}$ la propiedad de corrección que exigíamos en la sección 1.1.2? La respuesta es que sí:

Teorema 1.2.1. Sea $n = pq$ donde p y q son primos distintos. Sean e y d enteros tales que $ed \equiv 1 \mod (p-1)(q-1)$. Entonces para todo $x \in \mathbb{Z}$, se tiene $x^{ed} \equiv x \mod n$.

Demostración. En primer lugar notemos que la hipótesis $ed \equiv 1 \mod (p-1)(q-1)$ significa que $ed = 1 + k(p-1)(q-1)$ para algún entero k.

Tomemos ahora $x \in \mathbb{Z}$ y probemos que $x^{ed} \equiv x \mod p$ y $x^{ed} \equiv x \mod q$. Esto nos dará que $x^{ed} - x$ es divisible por los primos distintos p y q, y por tanto debe ser divisible por su producto n, lo que significa que $x^{ed} \equiv x \mod n$.

En efecto, si $x \equiv 0 \mod p$, entonces es inmediato que $x^{ed} \equiv 0 \equiv x \mod p$. Por otro lado, en el caso de que $x \not\equiv 0 \mod p$, por el pequeño teorema de Fermat tenemos $x^{p-1} \equiv 1 \mod p$ y así,

$$x^{ed} \equiv x^{1+k(p-1)(q-1)} \equiv x \cdot \left(x^{p-1}\right)^{k(q-1)} \equiv x \cdot 1^{k(q-1)} \equiv x \mod p.$$

Hemos llegado pues a que $x^{ed} \equiv x \mod p$.

Por un argumento análogo, $x^{ed} \equiv x \mod q$. □

Ahora que sabemos que $\mathcal{T}_{RSA}$ es un esquema de permutación con trampa, cabe preguntarse si es de una vía. Recordando el juego en 1.1.2 donde definíamos el concepto de función de una vía con trampa, tenemos que $\mathcal{T}_{RSA}$ es de una vía si no hay un algoritmo eficiente tal que dados n y x^e, con $x \in \mathbb{Z}_n$ cualquiera, se pueda calcular x. Si $\mathcal{T}_{RSA}$ es de una vía, debe ser difícil factorizar n, ya que si factorizamos n, uno podría calcular d como en el algoritmo

$RSAGen$ y después computar $x = y^d$. Es ampliamente aceptado, por el desconocimiento de un método clásico efectivo que lo realice tras años de investigaciones, que si l está entre 1000 y 1500, entonces factorizar n es complicado. Sin embargo, no hay prueba de que el hecho de que sea difícil factorizar n implique que $\mathcal{T}_{RSA}$ sea de una vía. En la práctica se asume que esto es así, ya que hasta la fecha todas las evidencias apuntan a ello. Esta conjetura se formaliza mediante el siguiente juego:

Juego 1.2.1. Dados un número entero $l > 2$ y un número entero impar $e > 2$, y dado un adversario $\mathcal{A}$, el juego se desarrolla de la siguiente manera:

- El retador ejecuta $(n, d) \leftarrow RSAGen(l, e)$, toma un $x \in \mathbb{Z}_n$ y calcula $y = x^e \in \mathbb{Z}_n$.
- El retador envía (n, y) al adversario.
- El adversario elige un $\hat{x} \in \mathbb{Z}_n$.

Definimos la ventaja del adversario $\mathcal{A}$ en romper RSA, RSAadv$[\mathcal{A}, l, e]$ como la probabilidad de que $\hat{x} = x$.

Definición 1.2.2 (Suposición RSA). Diremos que la suposición RSA es válida para (l, e) si para todos los adversarios eficientes $\mathcal{A}$, la cantidad RSAadv$[\mathcal{A}, l, e]$ es despreciable.

1.2.2. Intercambio de claves usando RSA

Usando el procedimiento general de la sección 1.1.2 se puede considerar un protocolo anónimo de intercambio de claves basandonos en el esquema de permutación de una vía con trampa $\mathcal{T}_{RSA}$. Bajo la suposición RSA, será un protocolo anónimo de intercambio de claves seguro.

1.2.3. Criptosistema de clave pública RSA

Construiremos un criptosistema de clave pública CCA seguro basándonos en $\mathcal{T}_{RSA}$. Para ello, utilizaremos una función resumen modelada como un oráculo aleatorio y un cifrado simétrico (ver apéndice A para definiciones sobre cifrados simétricos).

Como ya hemos visto, el esquema de permutación con trampa RSA no se ajusta del todo a la definición de este concepto, ya que el dominio de las funciones cambia con la clave pública. Esto se puede solucionar tomando un conjunto $\mathcal{X}$ en el que podamos sumergir $\mathbb{Z}_n$ (por ejemplo $\{0,1\}^{2l}$). Usaremos un cifrado simétrico $\varepsilon_S = (E_S, D_S)$ definido sobre

$(\mathcal{K}, \mathcal{M}, \mathcal{C})$ y una función resumen $H : \mathcal{X} \longrightarrow \mathcal{K}$. Así, definimos el criptosistema de clave pública RSA $\varepsilon_{RSA} = (G, E, D)$ con espacio de mensajes $\mathcal{M}$ y espacio de mensajes cifrados $\mathcal{X} \times \mathcal{C}$, donde:

- El algoritmo $G(\cdot)$ hace:

$$(n, d) \leftarrow RSAGen(l, e), \quad pk = (n, e), \quad sk = (n, d), \quad \text{devolver } (pk, sk).$$

- Dada una clave pública $pk = (n, e)$ y un mensaje $m \in \mathcal{M}$, el algoritmo de cifrado hace:

  Tomar $x \in \mathbb{Z}_n$ aleatorio, $\quad y = x^e, \quad k = H(x), \quad c \leftarrow E_S(k, m), \quad$ devolver $(y, c) \in \mathcal{X} \times \mathcal{C}$.

- Dada una clave secreta $sk = (n, d)$ y un texto cifrado $(y, c) \in \mathcal{C} \times \mathcal{C}$, donde $y \in \mathbb{Z}_n$, el algoritmo de descifrado hace:

$$x = y^d, \quad k = H(x), \quad m = D_S(k, c), \quad \text{devolver } m.$$

A grandes rasgos, lo que hace este esquema de clave pública es cifrar mensajes con el cifrado simétrico, usando claves que se crean y se protegen por el RSA. Esta idea recibe el nombre de esquema 'híbrido' y la filosofía de su diseño es la que está detrás del concepto de KEM. Profundizaremos en ello en el capítulo 4.

¿Se tiene la propiedad de corrección deseada para un criptosistema de clave pública? Es fácil comprobar que sí. En efecto, dada una salida $(pk, sk) = ((n, e), (n, d))$ de $G(\cdot)$ y un mensaje $m \in \mathcal{M}$:

$$D(sk, E(pk, m)) = m \Longleftrightarrow D_S(H(y^d), E_S(H(x), m)) = m \Longleftrightarrow D_S(H(x^{ed}), E_S(H(x), m)) = m$$

Este último suceso tiene probabilidad 1 usando la propiedad de corrección del cifrado simétrico, pues debido al teorema 1.2.1 $x^{ed} = x$ (como elemento de $\mathbb{Z}_n$). Además, el siguiente teorema nos dice bajo qué condiciones este criptosistema es CCA seguro:

Teorema 1.2.2. Sea $H : \mathcal{X} \longrightarrow \mathcal{K}$ una función resumen modelada como un oráculo aleatorio. Si la suposición RSA es válida para (l, e), y ε_S es 1CCA seguro, entonces ε_{RSA} es CCA seguro.

1.2.4. Firma digital basada en el esquema de permutación con trampa RSA

A la hora de utilizar el esquema de permutación con trampa $\mathcal{T}_{RSA}$ para construir una firma digital, de nuevo nos encontramos con el problema de que el dominio de las funciones cambia con cada clave pública. Para utilizar una función resumen necesitamos tener un dominio y un espacio de llegada fijos. A tal fin, tomamos el conjunto $\mathcal{Y} := \{1, \ldots, 2^{2l-2}\}$ y consideramos una función resumen $H : \mathcal{M} \longrightarrow \mathcal{Y}$. El conjunto $\mathcal{Y}$, por construcción, se puede sumergir en $\mathbb{Z}_n$, para todos los n generados por $RSAGen(l, e)$. Definimos entonces el esquema de firma digital $\mathcal{S}_{RSA} = (G, S, V)$, donde los diferentes algoritmos se definen como:

- El algoritmo de generación de claves usa los parámetros l y e y hace:

$$(n, d) \leftarrow RSAGen(l, e), \quad pk = (n, e), \quad sk = (n, d), \quad \text{devolver } (pk, sk)$$

- El algoritmo de firma S, dada una clave secreta $sk = (n, d)$ y un mensaje $m \in \mathcal{M}$, hace:

$$y = H(m) \in \mathcal{Y}, \quad \sigma = y^d \in \mathbb{Z}_n, \quad \text{devolver } \sigma$$

- El algoritmo de verificación V, dada una clave pública $pk = (n, e)$, un mensaje $m \in \mathcal{M}$ y un $\sigma \in \mathbb{Z}_n$ hace:

$$y = \sigma^e \in \mathbb{Z}_n. \quad \text{Si } y = H(m), \text{ devolver } \texttt{accept}\text{; en otro caso, devolver } \texttt{reject}$$

Observación. La propiedad de que una firma generada por S sea siempre aceptada por V se tiene como consecuencia del teorema 1.2.1.

Observación. En este esquema de firma digital es muy importante usar una función resumen antes de firmar. A continuación se muestra como, en el caso de que la firma estuviese definida simplemente como $\sigma := m^d$, un atacante podría obtener una firma en el mensaje m que quisiese, sin más que pedir una firma de otro mensaje $\hat{m}$. El atacante haría:

Tomar aleatoriamente $r \in \mathbb{Z}_n^*$, $\quad \hat{m} = mr^e$.

Pedir la firma de $\hat{m}$, obteniendo así $\hat{\sigma} = \hat{m}^d$.

Calcular $\sigma = \dfrac{\hat{\sigma}}{r}$

Así, si $\hat{\sigma}^e = \hat{m}$, entonces $\sigma \in \mathbb{Z}_n$ y es una firma válida para m pues

$$\sigma^e = \left(\frac{\hat{\sigma}}{r}\right)^e = \frac{\hat{\sigma}^e}{r^e} = \frac{\hat{m}}{r^e} = m.$$

Vale la pena decir que esto es debido a que la transformación de firma es homomórfica. Se dice por tanto que las firmas digitales RSA sin función resumen son *universalmente falsificables* y, en consecuencia, nunca deben ser usadas.

Para la firma digital definida usando una función resumen, tenemos el siguiente resultado sobre su seguridad:

Teorema 1.2.3. Sea $H : \mathcal{M} \longrightarrow \mathcal{Y}$ una función resumen modelada como un oráculo aleatorio, con $\mathcal{Y} := \{1, \ldots, 2^{2l-2}\}$. Si la suposición RSA es válida para (l, e), entonces $\mathcal{S}_{RSA}$ con parámetros (l, e) es un esquema de firma digital seguro.

1.3. Criptografía basada en el problema del logaritmo discreto

Explicamos en esta sección la otra gran técnica para la criptografía de clave pública utilizada hoy en día, que es aquella basada en el problema del logaritmo discreto. En primer lugar, expondremos lo que es el problema del logaritmo discreto, y las suposiciones que se aceptan sobre su dificultad. Posteriormente lo utilizaremos para construir un protocolo anónimo de intercambio de claves, un criptosistema de clave pública CCA seguro, y un esquema de firma digital.

1.3.1. El problema del logaritmo discreto y suposiciones asociadas

En la discusión de este problema, nos restringiremos a grupos cíclicos $\mathbb{G}$ de orden primo q, tales que $\mathbb{G} =< g >$ para $g \in \mathbb{G}$. Estos grupos los construiremos como se detalla a continuación.

Sean p un primo grande (normalmente de 2048 bits) y q un primo algo menor (digamos de 256 bits) tal que $q \mid p-1$. Sabemos que en estas circunstancias $\mathbb{Z}_p^*$, el grupo multiplicativo de los restos no nulos módulo p, tiene un elemento g de orden q. Denotando $\mathbb{G} =< g >$, tenemos que para todo $u \in \mathbb{G}$, dados dos enteros a y b, $u^a = u^b \Longleftrightarrow a \equiv b \bmod q$. Así, el valor de u^a depende solo de la clase de a módulo q y si $\alpha = [a]_q$ podemos definir $u^\alpha := u^a$. Esto justifica que de ahora en adelante usemos elementos de $\mathbb{Z}_q$ como exponentes de elementos de $\mathbb{G}$.

En un grupo de estas características, dado un elemento $x \in \mathbb{G}$, el problema del logaritmo discreto consiste en encontrar el único entero α, con $0 \leq \alpha \leq q-1$, tal que $g^\alpha = x$. Como siempre, plantear este problema como un juego nos permitirá formalizar algunos conceptos importantes:

Juego 1.3.1 (Problema del logaritmo discreto). Sea $\mathbb{G}$ un grupo cíclico de orden q generado por un elemento $g \in \mathbb{G}$. Dado un adversario $\mathcal{A}$, se define el siguiente juego:

- El retador escoge un $\alpha \in \mathbb{Z}_q$, calcula $u = g^\alpha$ y se lo manda al adversario.
- El adversario elige un $\hat{\alpha} \in \mathbb{Z}_q$.

Definimos la ventaja de $\mathcal{A}$ al resolver el problema del logaritmo discreto para $\mathbb{G}$, denotado $\mathrm{DLadv}[\mathcal{A}, \mathbb{G}]$, como la probabilidad de que $\hat{\alpha} = \alpha$

Definición 1.3.1 (Suposición del logaritmo discreto). Decimos que la suposición del logaritmo discreto, abreviada como *suposición DL*, es válida para $\mathbb{G}$ si, para todos los adversarios eficientes $\mathcal{A}$, la cantidad $\mathrm{DLadv}[\mathcal{A}, \mathbb{G}]$ es despreciable.

Para construir nuestro protocolo de intercambio de claves necesitaremos hacer otra suposición, que formalizamos con el siguiente juego:

Juego 1.3.2 (Problema computacional de Diffie-Hellman). Sea $\mathbb{G}$ un grupo cíclico de orden primo q generado por un elemento $g \in \mathbb{G}$. Dado un adversario $\mathcal{A}$, el juego se desarrolla de la siguiente manera:

- El retador escoge α, β en $\mathbb{Z}_q$ y calcula $u = g^\alpha$, $v = g^\beta$ y $w = g^{\alpha\beta}$. Envía el par (u, v) al adversario.
- El adversario elige un $\hat{w} \in \mathbb{G}$.

Definimos la ventaja de $\mathcal{A}$ al resolver el problema computacional de Diffie-Hellman, $\mathrm{CDHadv}[\mathcal{A}, \mathbb{G}]$, como la probabilidad de que $\hat{w} = w$.

Definición 1.3.2. Diremos que (g^α, g^β) es una *instancia* del problema CDH. Por su parte, el elemento $(u, v, w) \in \mathbb{G}^3$ recibirá el nombre de tupla de Diffie-Hellman (tupla de DH).

Definición 1.3.3 (Suposición computacional de Diffie-Hellman). Diremos que la suposición computacional de Diffie-Hellman, abreviada como *suposición CDH*, es válida para $\mathbb{G}$ si para todos los adversarios eficientes $\mathcal{A}$, se tiene que $\mathrm{CDHadv}[\mathcal{A}, \mathbb{G}]$ es despreciable.

Observación. La suposición CDH implica la suposición DL. En efecto, si un adversario $\mathcal{A}$ pudiese resolver el problema del logaritmo discreto, al recibir u y v podría calcular α y β, obteniendo entonces $w = g^{\alpha\beta}$.

Observación. El problema computacional de Diffie-Hellman tiene la propiedad de que es imposible (salvo con probabilidad despreciable) reconocer si una tupla (u, v, w) es una tupla de DH. Esto da lugar a otra suposición, la decisional de Diffie-Hellman (DDH). Intuitivamente, un grupo cíclico de orden q $\mathbb{G}$ satisface la suposición DDH si dados $(u, v, w) \in \mathbb{G}^3$, es difícil saber si esta tupla es de DH. Notemos que la suposición DDH implica la suposición CDH.

1.3.2. Un protocolo de intercambio de claves

Pasamos pues a definir el protocolo de intercambio de claves basado en el problema del logaritmo discreto. Se trata de la propuesta original de Diffie y Hellman y es por tanto el primer esquema de clave pública conocido. Como se ha mencionado en la sección 1.1.2, esta propuesta no basa su funcionamiento en una función de una vía con trampa. En su lugar, suponiendo que todas las partes conocen una descripción de $\mathbb{G}$ donde se incluyen g y q, se procede de la siguiente manera:

- Alice elige aleatoria y uniformemente $\alpha \in \mathbb{Z}_q$, calcula $u = g^\alpha$ y se lo manda a Bob.
- Bob elige aleatoria y uniformemente $\beta \in \mathbb{Z}_q$, calcula $v = g^\beta$ y se lo manda a Alice.
- Tras recibir v de Bob, Alice calcula $w = v^\alpha$.
- Tras recibir u de Alice, Bob calcula $w = u^\beta$.

Al terminar el protocolo Alice y Bob comparten la clave $w = v^\alpha = g^{\alpha\beta} = u^\beta$.

Para primos p y q suficientemente grandes, se asume que se satisface la suposición DL. Sin embargo, esto no es suficiente para que este protocolo sea seguro, sino que necesitamos también la suposición CDH. Aunque esta suposición es más fuerte que la suposición DL, podemos asumir que en grupos donde la suposición DL se satisface tendremos también la suposición CDH. Al igual que con el problema de la factorización, esto se basa en evidencias tanto teóricas como computacionales.

1.3.3. El criptosistema de clave pública ElGamal

Propuesto en 1984, fue el primer criptosistema basado en el problema del logaritmo discreto, y aún hoy es el más famoso. Para construir este criptosistema, que denotaremos ε_{EG}, necesitaremos de nuevo un grupo cíclico $\mathbb{G}$ de orden primo q y generador $g \in \mathbb{G}$. Además necesitaremos un cifrado simétrico $\varepsilon_S = (E_S, D_S)$ definido sobre $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, y una función resumen $H : \mathbb{G}^2 \longrightarrow \mathcal{K}$. El espacio de mensajes en claro para ε_{EG} será $\mathcal{M}$ y el espacio de mensajes cifrados será $\mathbb{G} \times \mathcal{C}$. Los algoritmos de generación de claves, cifrado y descifrado $G(\cdot)$, $E(pk, m)$ y $D(sk, (v, c))$ vienen descritos a continuación:

- $G(\cdot) :=$

  Tomar aleatoria y uniformemente $\alpha \in \mathbb{Z}_q$, Calcular $u = g^\alpha$.

  Devolver $pk = u,\ sk = \alpha$.

- Dada una clave pública $pk = u \in \mathbb{G}$ y un mensaje $m \in \mathcal{M}$, para calcular $E(pk, m)$:

  Tomar aleatoria y uniformemente $\beta \in \mathbb{Z}_q$.

  Calcular $v = g^\beta,\ w = u^\beta,\ k = H(v, w),\ c = E_S(k, m)$.

  Devolver (v, c).

- Dada una clave privada $sk = \alpha \in \mathbb{Z}_q$ y un texto cifrado $(v, c) \in \mathbb{G} \times \mathcal{C}$, para calcular $D(sk, (v, c))$:

  Calcular $w = v^\alpha,\ k = H(v, w),\ m = D_S(k, c)$.

  Devolver m.

Al igual que con el RSA, en este esquema se cifra con el cifrado simétrico usando una clave protegida mediante el esquema de clave pública, dando lugar a un esquema híbrido.

Por otro lado, tenemos de nuevo un criptosistema que satisface la propiedad de corrección, esto es: dada una salida $(pk, sk) = (u, \alpha)$ de $G(\cdot)$ y un mensaje $m \in \mathcal{M}$:

$$D(sk, E(pk, m)) = m \Longleftrightarrow D_S(H(v, v^\alpha), E_S(H(v, u^\beta), m) = m.$$

Este último suceso se tiene con probabilidad 1 pues, por construcción, $v^\alpha = g^{\alpha\beta} = u^\beta$, pudiendo entonces usar la propiedad de corrección para ε_S.

Para probar resultados de CCA seguridad para este criptosistema, necesitaremos introducir otra suposición, la suposición CDH interactiva. Esto es necesario pues para la CCA seguridad implica que un adversario puede hacerle al retador preguntas de texto cifrado escogido. Si solo se verifican las suposiciones CDH y DDH, un adversario podría utlizar estas preguntas para obtener una ventaja no despreciable en el juego de la CCA seguridad de la siguiente manera:

Si la clave pública es $u = g^{\alpha}$, el adversario elige dos elementos arbitrarios $(\hat{v}, \hat{w}) \in \mathbb{G}^2$, y calcula $\hat{k} = H(\hat{v}, \hat{w})$ y $\hat{c} = E_S(\hat{k}, \hat{m})$, para algún mensaje $\hat{m}$. Después, haciendo uso de una pregunta de descifrado, le pide al retador que le calcule el descifrado m^* de $(\hat{v}, \hat{c})$. Ahora, es muy probable que $\hat{m} = m^*$ si y solo si $\hat{w} = \hat{v}^{\alpha}$, es decir, si y solo si $(u, \hat{v}, \hat{w})$ es una tupla de DH. El adversario puede pues figurarse la respuesta de la pregunta '¿es $(u, \hat{v}, \hat{w})$ una tupla de DH?', en cuyo caso tendría una pista sobre la clave privada. La suposición DDH implica que el adversario no puede responder por sí mismo a esta pregunta, pero ahora, usando las preguntas de texto cifrado escogido, ha adquirido esta capacidad. Debemos pues recogerlo en otra suposición, la suposición CDH interactivo (ICDH).

Intuitivamente, esta suposición dice que dada una instancia cualquiera (g^{α}, g^{β}) del problema CDH, es complicado calcular $g^{\alpha\beta}$ incluso con acceso a un 'oráculo de decisión de DH', esto es, un oráculo que reconoce si una tupla $(u, v, w) \in \mathbb{G}^3$ es una tupla de DH. Una vez esbozado este concepto, podemos citar el siguiente teorema de CCA seguridad para ε_{EG}:

Teorema 1.3.1. Supongamos que $H : \mathbb{G}^2 \longrightarrow \mathcal{K}$ esté modelada como un oráculo aleatorio. Si la suposición ICDH es cierto para $\mathbb{G}$ y ε_S es 1CCA seguro, entonces ε_{EG} es CCA seguro.

1.3.4. Firma digital basada en el problema del logaritmo discreto

En 1985, Taher Elgamal propuso un esquema de firma digital cuya seguridad estaba basada en el problema del logaritmo discreto. Sin embargo, en este esquema era posible hacer falsificaciones existenciales. Otro esquema de firma digital basado en el problema del logaritmo discreto, y que sí resulta ser UF-CMA seguro es el esquema de Schnorr. Este esquema fue propuesto en 1989 y se conoce una prueba a su seguridad desde 1996 ([4]). Veámoslo:

Sea $\mathbb{G}$ un grupo cíclico de orden primo q y sea g un generador de $\mathbb{G}$. Sea $H : \{0,1\}^* \times \mathbb{G} \longrightarrow \mathbb{Z}_q$ una función resumen[1]. Entonces:

[1] $\{0,1\}^*$ representa el conjunto de las cadenas de ceros y unos de longitud arbitraria.

- El algoritmo de generación de claves $G(\cdot)$ hace:

  1. Escoger de manera aleatoria y uniforme $x \in \mathbb{Z}_q \setminus \{0\}$.
  2. Calcular $y = g^x$.
  3. Devolver $(pk = y, sk = x)$.

- Dado un mensaje $m \in \{0,1\}^*$, el algoritmo de firma $S(sk, m)$ hace:

  1. Escoger de manera aleatoria y uniforme $a \in \mathbb{Z}_q$.
  2. Calcular $r = g^a$, $c = H(m||r)$ y[2] $s = a + cx \bmod q$.
  3. Devolver la firma (s, c).

- El algoritmo de verificación $V(pk, m, \sigma = (s, c))$ hace:

  1. Calcular $r = g^s y^{-c}$.
  2. Comprobar si $c = H(m||r)$.

La corrección de este esquema es inmediata teniendo en cuenta que si $\sigma = (s, c)$ es una firma válida de m, entonces $s = a + cx \bmod q$. De aquí se sigue que $s = a + cx + tq$ para algún entero t, y por tanto el r calculado en el algoritmo de verificación coincide con g^a. En efecto:

$$g^s y^{-c} = g^a g^{cx} g^{tq} g^{-cx} = g^a.$$

Además, tenemos el siguiente teorema sobre su seguridad:

Teorema 1.3.2. Si la suposición DL es válida para $\mathbb{G}$ y H es un oráculo aleatorio, entonces el esquema de firma digital de Schnorr es UF-CMA seguro.

Resumen del capítulo

Hemos visto las principales definiciones y resultados sobre la criptografía de clave pública, quedando familiarizados con la manera de hacer argumentos teóricos en base a 'juegos'. Asimismo, hemos introducido las técnicas criptográficas más utilizadas hoy en día. Esto motiva el siguiente capítulo, donde veremos cómo estas técnicas quedarán obsoletas con la llegada de los ordenadores cuánticos.

[2] El símbolo || denota la concatenación de cadenas de bits.

Capítulo 2

Computación cuántica. El algoritmo de Shor

2.1. ¿Qué es un ordenador cuántico?

En pocas palabras, un ordenador cuántico es un dispositivo de cálculo que en su funcionamiento aprovecha las leyes de la física cuántica. Actualmente tales ordenadores han sido desarrollados solo a tamaño muy reducido, ya que su construcción plantea enormes dificultades, entre las que destaca la necesidad de aislar los sistemas físicos que se manejan. Si uno de estos sistemas físicos interacciona con una molécula de aire, o absorbe una minúscula cantidad de energía del ambiente, el resultado de la computación que estemos efectuando puede ser inútil. Esta es una diferencia muy grande con los ordenadores normales (que llamaremos *clásicos*), donde en principio los diferentes componentes electrónicos están en contacto con el ambiente que los rodea.

Si es tan difícil, ¿por qué empeñarnos entonces en construir ordenadores cuánticos? Porque estos ordenadores tendrán una eficiencia para resolver determinados problemas enormemente mayor que la que jamás podrá alcanzar un ordenador clásico. Esta eficiencia es la que usa el algoritmo cuántico de Shor, que permitiría factorizar números enteros y resolver el problema del logaritmo discreto en tiempos asequibles. Por tanto, cuando se desarrolle a escala suficientemente grande un ordenador cuántico, las suposiciones RSA y DL expuestos en el capítulo anterior dejarán de ser válidas, y se hará imperiosa la necesidad de la criptografía postcuántica.

En este capítulo, se hará una introducción al proceso de cómputo en los ordenadores cuánticos, siguiendo lo expuesto en [5]. Posteriormente, se describirá el algoritmo cuántico de Shor que permite factorizar números enteros, también con la línea de [5], y el algoritmo de Shor para resolver el problema del logaritmo discreto, siguiendo en este caso [6].

2.2. Cbits y Qbits

Un ordenador clásico opera sobre cadenas de ceros y unos, del estilo de 1010011. Cada posición de esta cadena recibe el nombre de bit, y está representada por un sistema físico que puede estar en dos estados, uno que representa el 0 y otro que representa el 1. Siguiendo la notación de [5], llamaré Cbit a estos bits clásicos y Qbit al análogo de este concepto para ordenadores cuánticos. Además, representamos el estado de cada Cbit o de cada Qbit con la notación de Dirac, esto es, entre los símbolos $|\cdot\rangle$. Así, los dos estados diferenciados de los Cbits serán $|0\rangle$ y $|1\rangle$. Cuando tengamos un sistema con varios Cbits, lo denotaremos concatenando los diferentes kets, teniendo cadenas como

$$|1\rangle\,|0\rangle\,|1\rangle\,.$$

Estas cadenas serán representadas normalmente como $|101\rangle$ (poner todos los unos y los ceros en un solo ket) ó $|5\rangle$ (el entero cuya expansión en binario es la cadena de bits). Notemos que esta última notación es ambigua, pues $|5\rangle$ puede representar estados con diferente número de bits. Esta ambigüedad se resuelve, cuando es necesario, poniendo un subíndice que indique el número de bits involucrados, por ejemplo $|5\rangle_3$.

La notación de Dirac se utiliza en realidad para representar vectores. Aunque esto no es necesario para Cbits, será fundamental al tratar con Qbits. En general, los dos estados de un solo Cbit se representan por dos vectores ortogonales en un espacio hermítico de dimensión 2. En términos de componentes:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad\qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Cuando tengamos sistemas con varios Cbits, la notación $|1\rangle\,|0\rangle\,|1\rangle$ representa en realidad el

producto tensorial $|1\rangle \otimes |0\rangle \otimes |1\rangle$. En términos de componentes, esto significa:

$$\begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \otimes \begin{pmatrix} y_0 \\ y_1 \end{pmatrix} = \begin{pmatrix} x_0 y_0 \\ x_0 y_1 \\ x_1 y_0 \\ x_1 y_1 \end{pmatrix}.$$

La cosa cambia sustancialmente para los Qbits. Mientras que un Cbit solo puede estar en dos estados bien definidos, el estado de un Qbit, que podemos denotar $|\psi\rangle$, será en general una combinación lineal compleja

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle = \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix}$$

donde los coeficientes están sujetos a la condición de normalización $|\alpha_0|^2 + |\alpha_1|^2 = 1$. Se dice en este caso que el estado $|\psi\rangle$ es una superposición de los estados $|0\rangle$ y $|1\rangle$ con amplitudes α_0 y α_1.

En el caso de tener un sistema con n Qbits, el estado del sistema $|\Psi\rangle$ está en un espacio hermítico de dimensión 2^n, de modo que en general podrá ser escrito en la forma

$$|\Psi\rangle = \sum_{0 \le x < 2^n} \alpha_x |x\rangle_n , \qquad \text{con} \qquad \sum_{0 \le x < 2^n} |\alpha_x|^2 = 1 \tag{2.1}$$

donde, como en el caso clásico, $|x\rangle_n$ representa el producto tensorial de n estados $|0\rangle$ ó $|1\rangle$ para un Qbit individual, tales que la cadena en binario representa el número decimal x. Esta base del espacio se llama *base clásica* o *base computacional*.

A su vez, si tenemos dos Qbits, uno en el estado $|\psi\rangle = \beta_0 |0\rangle + \beta_1 |1\rangle$ y otro en el estado $|\varphi\rangle = \gamma_0 |0\rangle + \gamma_1 |1\rangle$, el par estará descrito por el producto tensorial

$$\begin{aligned} |\Psi\rangle = |\psi\rangle \otimes |\varphi\rangle &= (\beta_0 |0\rangle + \beta_1 |1\rangle) \otimes (\gamma_0 |0\rangle + \gamma_1 |1\rangle) \\ &= \beta_0\gamma_0 |0\rangle_2 + \beta_0\gamma_1 |1\rangle_2 + \beta_1\gamma_0 |2\rangle_2 + \beta_1\gamma_1 |3\rangle_2 . \end{aligned}$$

Como indica esta expresión, un producto tensorial de estados de dos Qbits puede ser expresado en la forma de (2.1), pero no cualquier estado de dos Qbits en la forma de (2.1) podrá ser expresado como producto tensorial de estados de dos Qbits individuales. En concreto, para un estado $|\Xi\rangle = \alpha_0 |0\rangle_2 + \alpha_1 |1\rangle_2 + \alpha_2 |2\rangle_2 + \alpha_3 |3\rangle_2$ esto solo sucederá si y solo si $\alpha_0\alpha_3 = \alpha_1\alpha_2$. Este fenómeno, por el que el estado de un sistema de 2 Qbits no puede descomponerse como el producto de los estados de los Qbits individuales, se conoce como

entrelazamiento cuántico y marca una diferencia enorme con el caso clásico, donde un sistema de varios Cbits está siempre en un estado que es producto de los estados de los Cbits individuales.

2.3. Operaciones reversibles sobre Qbits

Las operaciones con las que funcionan los ordenadores cuánticos son operaciones reversibles; es decir, transformaciones que llevan los Qbits de un estado inicial a uno final, utilizando procesos que puede ser revertidos. De hecho, los ordenadores cuánticos solo utilizan una transformación irreversible, la *medida*, que será por otra parte la única manera de extraer información útil de los Qbits. Veremos más adelante cómo se efectúa esta medida, y las cruciales diferencias de concepto que hay con el caso clásico.

La operación reversible más general que puede hacer un ordenador cuántico a un sistema de n Qbits se representa con la acción sobre su estado de una transformación unitaria $\mathbf{U}$, esto es una transformación lineal que lleva vectores unitarios en vectores unitarios. Es conocido que $\mathbf{U}$ debe satisfacer que $\mathbf{U}\mathbf{U}^\dagger = \mathbf{U}^\dagger\mathbf{U} = 1$, donde $\dagger$ denota la transformación conjugada.

En la práctica, debido a dificultades técnicas, las transformaciones unitarias que se podrán realizar serán aquellas que puedan expresarse como producto de transformaciones unitarias sobre 1 ó 2 Qbits (llamadas respectivamente puertas de 1 Qbit o de 2 Qbits). Todos los algoritmos tratados en esta sección pueden ser implementados con puertas de 1 y 2 Qbits.

Una transformación unitaria que será de vital importancia es la llamada *transformación de Hadamard*. Como hemos visto, el estado de un Qbit está en un espacio de base $\{|0\rangle, |1\rangle\}$. En esta base, la transformación de Hadamard viene dada por

$$\mathbf{H} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

de modo que $\mathbf{H}|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ y $\mathbf{H}|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$.

2.4. Puertas de medida y la regla de Born

Para conocer el estado de un Cbit, basta mirarlo y ver si está en el estado $|0\rangle$ o en el estado $|1\rangle$, siendo éste un proceso que no altera el estado en que está el Cbit. En cambio, si

tenemos un Qbit en un estado $|\varphi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$, no hay ningún modo mediante el cual podamos conocer cuál es el estado del Qbit; es decir, no podemos conocer las amplitudes α_0 y α_1. Lo único que podemos hacer es aplicarle una *puerta de medida*, que además es un proceso que cambia irreversiblemente el estado del Qbit.

Una puerta de medida es un proceso que, aplicado a un Qbit, nos da el resultado $|0\rangle$ o $|1\rangle$. El hecho de obtener un cero o un uno no está determinado en general por el estado del Qbit, sino que lo único que este estado determina es la probabilidad de obtener un resultado u otro. Esta probabilidad está dada por la *regla de Born*, según la cual la probabilidad p_i de obtener la medida $i \in \{0, 1\}$ es $|\alpha_i|^2$.

En el caso en que tengamos un estado descrito por el producto tensorial de n Qbits, como en (2.1), lo que haremos será aplicarle la puerta de medida a cada Qbit individual. Obtendremos una colección de ceros y unos que no está determinada por el estado, sino que una vez más éste sólo determina la probabilidad de obtener una colección u otra. La regla de Born establece que dado un estado como en (2.1), la probabilidad de que los ceros y unos resultantes de la medida de todos los Qbits dé la expresión binaria del entero x es $p(x) = |\alpha_x|^2$. Como indicamos al inicio de esta sección, las puertas de medida no son reversibles: dado el estado final $|x\rangle_n$, el estado inicial no puede ser reconstruido. De hecho, una puerta de medida ni siquiera es lineal.

Otro aspecto muy importante de las puertas de medida es el llamado *colapso del estado*. Si a un estado $|\Psi\rangle$ descrito por el producto tensorial de n Qbits se le aplica una puerta de medida, obteniéndose el valor x, entonces el estado después de la medida es $|x\rangle_n$. De este modo, después de mandar el estado $|\Psi\rangle$ a través de una puerta de medida, ya no tenemos más oportunidades para obtener más información acerca de él. ¿Cómo podemos obtener información de interés entonces a partir de los Qbits? La idea es, aplicando transformaciones unitarias apropiadas, obtener estados donde todas las amplitudes α_x salvo unas pocas sean 0 o muy cercanas a 0. La información útil estará contenida en cualquiera de los resultados que tienen una probabilidad no nula de ser obtenidos.

La regla de Born admite un generalización, conocida como *regla de Born generalizada*. Partimos de un estado descrito por el producto tensorial de $m+n$ Qbits, que se puede poner en su forma general como

$$|\Psi\rangle_{m+n} = \sum_{x=0}^{2^m-1} \alpha_x |x\rangle_m |\Phi_x\rangle_n ,$$

con $\sum_x |\alpha_x|^2 = 1$ y estados $|\Phi_x\rangle$ normalizados pero no necesariamente ortogonales. La regla dice que si uno mide solamente los m Qbits de la izquierda, entonces con probabilidad $|\alpha_x|^2$ el resultado será $|x\rangle$, y que en este caso después de la medida el estado será $|x\rangle_m |\Phi_x\rangle_n$.

2.5. Proceso computacional general en un ordenador cuántico

En general, un programa en un ordenador cuántico debe actuar sobre un número x para darnos otro número $f(x)$, donde f será la función que deseemos. Si disponemos de k Qbits, podremos representar en la base computacional enteros menores que 2^k. En concreto, si el número x es de n bits, y la salida $f(x)$ es de m bits, necesitaremos una colección de $n+m$ Qbits: n Qbits para representar x en lo que llamamos el *registro de entrada* o *input*, y m Qbits para representar $f(x)$ en el llamado *registro de salida* o *output*. Se puede ver el cálculo de f simplemente como aplicar una transformación unitaria $\mathbf{U}_f$ a los $n+m$ Qbits. El protocolo estándar de computación cuántica define esta transformación en la base computacional[1] $|x\rangle_n |y\rangle_m$ del siguiente modo:

$$\mathbf{U}_f \left(|x\rangle_n |y\rangle_m\right) = |x\rangle_n |y \oplus f(x)\rangle_m$$

donde $\oplus$ representa la suma módulo 2 bit a bit. Esta transformación es invertible (de hecho, unitaria), pues $\mathbf{U}_f$ es inverso de sí mismo. En efecto:

$$\mathbf{U}_f \mathbf{U}_f \left(|x\rangle |y\rangle\right) = \mathbf{U}_f(|x\rangle |y \oplus f(x)\rangle) = |x\rangle |y \oplus f(x) \oplus f(x)\rangle = |x\rangle |y\rangle$$

Por otra parte, si el valor inicial en el output es $y = 0$, entonces

$$\mathbf{U}_f \left(|x\rangle_n |0\rangle_m\right) = |x\rangle_n |f(x)\rangle_m ,$$

lo que explica la codificación de la función f a través de la matriz U_f. Además, esto sugiere uno de los trucos más importantes de la computación cuántica, que es de gran utilidad para comprender la enorme ventaja computacional que supone frente a la computación clásica. Para un operador $\mathbf{U}$ definimos su producto tensorial

$$\mathbf{U}^{\otimes n} = \mathbf{U} \otimes \mathbf{U} \otimes \cdots \otimes \mathbf{U}, \quad n \text{ veces}$$

[1] Por construcción, el producto tensorial de bases computacionales es otra base computacional.

Tomando entonces $\mathbf{H}$ la transformación de Hadamard, tenemos que

$$\mathbf{H}^{\otimes n} |0\rangle_n = \frac{1}{2^{\frac{n}{2}}} \sum_{0 \leq x < 2^n} |x\rangle_n$$

Esto puede ser probado mediante un argumento inductivo. Para $n = 1$, se trata de la definición dada para la transformación de Hadamard. Después, asumiendo que es cierto hasta $n-1$, se prueba para n:

$$\begin{aligned} \mathbf{H}^{\otimes n} |0\rangle_n &= \left(\mathbf{H}^{\otimes n-1} |0\rangle_{n-1}\right) \left(\mathbf{H} |0\rangle_1\right) \\ &= \frac{1}{2^{\frac{n}{2}}} \left(\sum_{0 \leq x < 2^{n-1}} |x\rangle_n \right) \left(|0\rangle_1 + |1\rangle_1\right) = \frac{1}{2^{\frac{n}{2}}} \sum_{0 \leq x < 2^n} |x\rangle_n . \end{aligned}$$

Esto quiere decir que si el estado del input es $|0\rangle_n$ (algo que se puede construir en la práctica) y le aplicamos $\mathbf{H}^{\otimes n}$, obtenemos un estado que es una combinación donde todos los estados posibles con n Qbits tienen el mismo peso. Aplicando entonces $\mathbf{U}_f$, con $|0\rangle_m$ en el output, obtenemos:

$$\begin{aligned} \mathbf{U}_f(\mathbf{H}^{\otimes n} \otimes \mathbf{1}_m)(|0\rangle_n |0\rangle_m) &= \frac{1}{2^{\frac{n}{2}}} \sum_{0 \leq x < 2^n} \mathbf{U}_f(|x\rangle_n |0\rangle_m) \\ &= \frac{1}{2^{\frac{n}{2}}} \sum_{0 \leq x < 2^n} |x\rangle_n |f(x)\rangle_m . \end{aligned}$$

De este modo, simplemente aplicando estas dos transformaciones, el estado final de esta transformación contiene 2^n evaluaciones de la función f. Este aparente milagro recibe el nombre de *pararelismo cuántico*. Debemos notar no obstante que, aunque tengamos un estado que contiene 2^n evaluaciones de f, en realidad no podemos conocer todas estas evaluaciones. Si hiciéramos una medida, que como sabemos es la única forma de extraer información de la computación cuántica, el estado de los registros de entrada y salida colapsaría a un $|x_0\rangle |f(x_0)\rangle$, para algún x_0, de modo que en la práctica solo obtendríamos este valor.

En este punto vale la pena mencionar el *teorema de no clonación*. Si hubiese una forma de hacer copias de un estado antes de hacer la medida, podríamos obtener tantos pares x_0, $f(x_0)$ como copias fuéramos capaces de hacer. Sin embargo, el teorema de no clonación establece que este proceso de copia no puede ser realizado. Se enuncia formalmente y demuestra a continuación:

Teorema 2.5.1 (Teorema de no clonación). No existe una transformación unitaria que lleve $|\psi\rangle_n |0\rangle_n$ a $|\psi\rangle_n |\psi\rangle_n$, para todo estado arbitrario $|\psi\rangle_n$.

Demostración. Es una consecuencia de la linearidad de los operadores.

Supongamos que existiese un operador lineal $\mathbf{U}$ tal que $\mathbf{U}(|\psi\rangle_n |0\rangle_n) = |\psi\rangle_n |\psi\rangle_n$ y $\mathbf{U}(|\phi\rangle_n |0\rangle_n) = |\phi\rangle_n |\phi\rangle_n$, para estados $|\psi\rangle$ y $|\phi\rangle$ arbitrarios. Entonces, por linearidad,

$$\mathbf{U}\left((a|\psi\rangle + b|\phi\rangle)|0\rangle\right) = a\mathbf{U}|\psi\rangle|0\rangle + b\mathbf{U}|\phi\rangle|0\rangle = a|\psi\rangle|\psi\rangle + b|\phi\rangle|\phi\rangle.$$

Pero $\mathbf{U}$ clona estados arbitrarios, de modo que

$$\mathbf{U}\left((a|\psi\rangle + b|\phi\rangle)|0\rangle\right) = (a|\psi\rangle + b|\phi\rangle)(a|\psi\rangle + b|\phi\rangle) = a^2|\psi\rangle|\psi\rangle + b^2|\phi\rangle|\phi\rangle + ab|\psi\rangle|\phi\rangle + ba|\phi\rangle|\psi\rangle.$$

Estas dos expresiones son diferentes, salvo en el caso en que a y b sean 0. Por tanto no puede existir un operador lineal en tales condiciones. □

Así, no es del todo cierto que hayamos obtenido 2^n evaluaciones de f, no al menos accesibles independientemente. En cualquier caso, aplicando ciertas transformaciones seremos capaces de obtener información que con un ordenador clásico sería costosísimo obtener.

2.6. El algoritmo de Shor para RSA

Ahora que hemos visto el funcionamiento general de los ordenadores cuánticos, veamos cómo funciona el algoritmo cuántico de Shor para romper la suposición RSA.

Como de costumbre, nos enfrentamos a la situación en que Bob quiere que Alice le mande un mensaje que sea secreto para todos salvo ellos dos. Para ello, como hemos visto, escoge un impar $e > 2$ y un entero $l > 2$ y ejecuta el algoritmo $RSAGen(l, e)$. Después publica $n = pq$ y e como claves públicas, donde por construcción e no tendrá ningún factor en común con $(p-1)(q-1)$. Alice toma estas claves públicas y, para enviarle a Bob el mensaje $a \in \mathbb{Z}_n$, calcula $b \equiv a^e \bmod n$ y se lo manda a Bob.

Lo que hace en realidad el algoritmo de Shor es encontrar el periodo de una función. Un adversario que tenga un algoritmo eficiente para calcular periodos de funciones puede utilizarlo para factorizar n, y como hemos visto esta capacidad le permitiría calcular la clave secreta, pero también puede utilizarlo directamente para obtener el mensaje a. Para explicar cómo, conviene definir el conjunto G_n, dado por todos los enteros positivos menores que n y que no tienen factores en común con él, es decir, el conjunto de unidades módulo n. Como se vio en Álgebra I, $(G_n, \cdot)$, donde $\cdot$ es la multiplicación módulo n, es un grupo.

En estas circunstancias, el adversario Eve utilizará el algoritmo para encontrar periodos para encontrar el periodo de la función $f(x) = b^x \bmod n$, es decir, el menor entero r para

el cual $b^{r+1} \equiv b \bmod n$. En el caso que $b \in G_{pq}$, esto no es otra cosa que el orden de b en este grupo.[2]

Además, este orden r debe ser el mismo que el de a. En efecto, el subgrupo de G_{pq} generado por a contiene a $b = a^e$, y por tanto contiene al subgrupo generado por b. Recíprocamente, el subgrupo generado por b contiene a $a = b^d$, con d la clave secreta, y por tanto contiene al subgrupo generado por a. Ambos grupos deben por tanto ser idénticos, y como el orden de un elemento es el número de elementos del subgrupo que genera, tenemos que ambos órdenes son iguales. Así, el adversario Eve conoce el orden r de a.

Por construcción, e no tiene factores en común con $(p-1)(q-1)$. Además r divide a $(p-1)(q-1)$, pues r es el orden de un subgrupo de G_{pq}, y el orden de G_{pq} es $\varphi(pq) = \varphi(p)\varphi(q) = (p-1)(q-1)$, con φ la función φ de Euler. En consecuencia, e no puede tener factores en común con r. Así, e es congruente módulo r a un elemento e' de G_r, que tendrá un inverso d' en el grupo G_r. Es inmediato que entonces se verifica $ed' \equiv 1 \bmod r$. Como Eve dispone de e y n, que son públicos, y de r, puede calcular d' usando el mismo procedimiento con el que Bob calculó $d \equiv e^{-1} \bmod (p-1)(q-1)$. Entonces, para algún entero m:

$$b^{d'} \equiv a^{ed'} \bmod pq \equiv a^{1+mr} \bmod pq \equiv a(a^r)^m \bmod pq \equiv a \bmod pq.$$

De este modo, Eve ha interceptado el mensaje a.

2.6.1. Algoritmo de Shor. Primera parte.

Hemos visto que encontrar el periodo de $f(x) = b^x \bmod pq$ supone romper mensajes en el criptosistema RSA. Recordemos que clásicamente los algoritmos que se conocen para encontrar el periodo no son eficientes, así que se necesita un algoritmo cuántico, que describiremos a continuación siguiendo lo expuesto en [5].

Sea n_0 el número de bits en la representación binaria de $n = pq$ (típicamente n será de 500 cifras, así que $n_0 \approx 1700$). Utilizaremos un registro de entrada de $N = 2n_0$ Qbits y un registro de salida de n_0 Qbits. Aplicando los operadores $\mathbf{H}^{\otimes n}$ y $\mathbf{U}_f$ como hemos visto,

[2]Notemos que en el caso de que $b \notin G_{pq}$, se puede aplicar el algoritmo de Euclides para encontrar $mcd(b, pq) \neq 1$, obteniendo la factorización de n y rompiendo RSA. En cualquier caso, debido al gran tamaño de los primos p y q, esto no deja de ser un caso anecdótico.

ponemos los registros de entrada y salida en el estado

$$\frac{1}{2^{\frac{N}{2}}} \sum_{x=0}^{2^N-1} |x\rangle_N |f(x)\rangle_{n_0} .$$

En este momento hacemos una medida sobre los n_0 bits del registro de salida. Si la medida nos da el valor f_0, la regla de Born generalizada nos asegura que el estado del registro de entrada es

$$|\Psi\rangle_N = \frac{1}{\sqrt{m}} \sum_{k=0}^{m-1} |x_0 + kr\rangle_N , \tag{2.2}$$

donde r es el periodo de f, $x_0 \in [0, r)$ es el menor valor para el que $f(x_0) = f_0$, y m es el menor entero para el que $mr + x_0 \geq 2^N$, es decir $m = \left[\frac{2^N}{r}\right]$ ó $m = \left[\frac{2^N}{r}\right] + 1$.

Es interesante darse cuenta de que si pudiésemos hacer copias del estado (2.2), entonces habríamos resuelto el problema. Haciendo diferentes medidas, obtendríamos diferentes valores $x_0 + k_i r$. Restándolos obtendríamos diferentes múltiplos de r, de donde sería sencillo obtener el valor de r. Sin embargo, esta opción no es posible debido al teorema de no clonación. Todo lo que podríamos obtener midiendo el estado (2.2) es un único valor $x_0 + kr$, inútil por sí solo para determinar r. Si ejecutásemos de nuevo el algoritmo, obtendríamos otro valor de f_0 y por tanto otro x_0, y no habría forma de relacionar los resultados de las dos ejecuciones para obtener r. Debemos por tanto idear algo más ingenioso. Este 'algo' es la *transformada cuántica de Fourier*, que nos permitirá llevar la dependencia de x_0 a una fase que multiplica a todo el estado.

2.6.2. La transformada cuántica de Fourier

La transformada cuántica de Fourier sobre N Qbits se define como la transformación $\mathbf{U}_{FT}$ que a cada estado $|x\rangle_N$ de la base computacional le asocia:

$$\mathbf{U}_{FT} |x\rangle_N = \frac{1}{2^{N/2}} \sum_{y=0}^{2^N-1} e^{2\pi i xy/2^N} |y\rangle_N ,$$

siendo xy la multiplicación ordinaria. Podemos probar que $\mathbf{U}_{FT}$ es una transformación unitaria. Usando la notación de Dirac, basta probar que en la base computacional se verifica $\langle x'| \mathbf{U}_{FT}^\dagger \mathbf{U}_{FT} |x\rangle = \delta_{xx'}$. Demuestro este resultado en el siguiente lema:

Lema. Sea $|x\rangle_N$, con $x = 0, \ldots, 2^N - 1$ la base computacional de un sistema de N Qbits. Entonces, para cualesquiera $x, x' \in \{0, \ldots, 2^N - 1\}$, se verifica que $\langle x'| \mathbf{U}_{FT}^\dagger \mathbf{U}_{FT} |x\rangle = \delta_{xx'}$

Demostración. Sean $x, x' \in \{0, \ldots, 2^N - 1\}$. Entonces

$$\left\langle \mathbf{U}_{FT}\, x' \middle| \mathbf{U}_{FT}\, x \right\rangle = \frac{1}{2^N} \left\langle \sum_{y=0}^{2^N-1} e^{\frac{2\pi i x' y}{2^N}} y \middle| \sum_{z=0}^{2^N-1} e^{\frac{2\pi i x z}{2^N}} z \right\rangle = \frac{1}{2^N} \sum_{y,z=0}^{2^N-1} e^{\frac{2\pi i (xz - x'y)}{2^N}} \langle y|z\rangle$$
$$= \frac{1}{2^N} \sum_{y,z=0}^{2^N-1} e^{\frac{2\pi i (xz - x'y)}{2^N}} \delta_{yz} = \frac{1}{2^N} \sum_{y=0}^{2^N-1} e^{\frac{2\pi i y (x - x')}{2^N}}.$$

- Si $x = x'$, entonces

$$\langle \mathbf{U}_{FT}\, x | \mathbf{U}_{FT}\, x \rangle = \frac{1}{2^N} \sum_{y=0}^{2^N-1} e^{\frac{2\pi i y (0)}{2^N}} = \frac{1}{2^N} \sum_{y=0}^{2^N-1} 1 = 1.$$

- Si $x \neq x'$, escribo $x - x' = 2^k t$, con t impar; es decir, meto todos los factores 2 de la factorización de $x - x'$ en 2^k (k puede ser 0, en cuyo caso $x - x'$ sería impar). En estas circunstancias, descompongo el sumatorio somo sigue:

$$\left\langle \mathbf{U}_{FT}\, x' \middle| \mathbf{U}_{FT}\, x \right\rangle = \frac{1}{2^N} \sum_{y=0}^{2^N-1} e^{\frac{2\pi i y (x - x')}{2^N}} = \frac{1}{2^N} \sum_{y=0}^{2^N-1} e^{\frac{2\pi i y 2^k t}{2^N}} =$$
$$= \frac{1}{2^N} \sum_{y=0}^{2^{N-1}-1} \left[e^{\frac{2\pi i 2^k t y}{2^N}} + e^{\frac{2\pi i 2^k t}{2^N}(y + 2^{N-k-1})} \right]$$
$$= \frac{1}{2^N} \sum_{y=0}^{2^{N-1}-1} \left[e^{\frac{2\pi i 2^k t y}{2^N}} + e^{\frac{2\pi i 2^k t y}{2^N}} e^{\pi i t} \right] = 0.$$

□

Por otro lado, se puede probar, y se hace en [5], que la transformada cuántica de Fourier se puede implementar con un número razonable puertas de uno y dos Qbits, de modo que será posible implementarla en un ordenador cuántico.

2.6.3. Algoritmo de Shor. Segunda parte

Volvamos a donde estábamos antes de introducir la transformada cuántica de Fourier, donde teníamos el registro de entrada en la forma (2.2). Aplicando $\mathbf{U}_{FT}$ a este registro de entrada, obtenemos:

$$\mathbf{U}_{FT} \frac{1}{\sqrt{m}} \sum_{k=0}^{m-1} |x_0 + kr\rangle = \frac{1}{2^{N/2}} \frac{1}{\sqrt{m}} \sum_{y=0}^{2^N-1} \sum_{k=0}^{m-1} e^{\frac{2\pi i (x_0 + kr) y}{2^N}} |y\rangle$$
$$= \sum_{y=0}^{2^N-1} e^{2\pi i x_0 y / 2^N} \frac{1}{\sqrt{2^N m}} \left(\sum_{k=0}^{m-1} e^{2\pi i k r y / 2^N} \right) |y\rangle .$$

Si ahora hacemos una medida, la probabilidad $p(y)$ de obtener el resultado $|y\rangle$ es, como de costumbre, el módulo del coeficiente que acompaña a $|y\rangle$ en esta expresión, es decir

$$p(y) = \frac{1}{2^N m}\left|\sum_{k=0}^{m-1} e^{2\pi i k r y/2^N}\right|^2. \tag{2.3}$$

Hemos conseguido por tanto algo que no depende del valor x_0, que era lo que nos molestaba para poder seguir adelante. ¿Cómo obtenemos ahora el valor de r a partir de (2.3)? El siguiente lema nos indica que al hacer una medida tenemos una gran probabilidad de que el resultado de la medida sea un y cercano a un múltiplo de $2^N/r$:

Lema. La probabilidad de obtener un resultado $|y\rangle$ que diste menos de $1/2$ de un múltiplo de $2^N/r$ es al menos 0,4.

Demostración. Fijado un $j \in \mathbb{Z}$, definimos $y_j = j2^N/r + \delta_j$, con $|\delta_j| \leq \frac{1}{2}$. Es inmediato que solo el término δ_j contribuye a las exponenciales en (2.3). La expresión es una serie geométrica, que podemos sumar. Aplicando algunas identidades trigonométricas:

$$p(y_j) = \frac{1}{2^N m}\frac{\sin^2(\pi\delta_j m r/2^N)}{\sin^2(\pi\delta_j r/2^N)}.$$

Como hemos visto, m se aleja como mucho en una unidad de $2^N/r$, lo que nos permite aproximar $mr/2^N \approx 1$. Además, como $N = 2n_0$, con n_0 el número de bits necesarios para representar $n = pq$, se tiene $2^N/r \geq n^2/r > n$, lo que indica que el argumento del seno del denominador es extremadamente pequeño. De este modo la probabilidad $p(y_j)$ queda:

$$p(y_j) = \frac{1}{r}\left(\frac{\sin(\pi\delta_j)}{\pi\delta_j}\right)^2.$$

Como $|\delta_j| < \frac{1}{2}$, se puede comprobar que

$$\frac{\sin(\pi\delta_j)}{\pi\delta_j} \geq \frac{2}{\pi},$$

de modo que

$$p(y_j) \geq (4/\pi^2)/r.$$

Como se puede elegir j de al menos $r-1$ formas posibles, y como r es un número grande (de otro modo no estaríamos aplicándole un algoritmo cuántico), tenemos que la probabilidad de obtener algún y_j es al menos 0,4; pues $4/\pi^2 \approx 0,4$. □

Supongamos por tanto que el resultado de la medida es $y = y_j$ para algún j. En estas circunstancias

$$\left| \frac{y}{2^N} - \frac{j}{r} \right| \leq \frac{1}{2^{N+1}}.$$

Como conocemos y y N, tenemos una estimación de la fracción j/r. En este punto se hace esencial haber elegido el registro de entrada con $N = 2n_0$ Qbits. Debido a esta elección $2^N > n^2$, de tal forma que nuestra estimación de j/r se aleja del valor exacto en menos de $1/(2n^2)$. Pero $r < n$, y se puede probar fácilmente que dos fracciones distintas de denominador menor que n difieren en al menos $1/n^2$. De este modo podemos deducir un valor exacto para el número racional j/r, utilizando la teoría de fracciones continuas. Veámoslo informalmente:

Definición 2.6.1. La expansión en fracciones continuas de un número real $x \in (0, 1]$ es

$$x = \cfrac{1}{a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \ldots}}}$$

Observación. a_0 es la parte entera de $1/x$. Si x_1 es la parte decimal de $1/x$, a_1 es la parte entera de $1/x_1$. Procediendo de este modo se obtienen los diferentes valores a_i.

Definición 2.6.2. Las sumas parciales de una fracción continua son los términos $\frac{1}{a_0}$, $\frac{1}{a_0 + \frac{1}{a_1}}$, etc.

Proposición 2.6.1. Si x es una estimación de j/r que difiere de él menos de $1/(2r^2)$, entonces j/r aparecerá como suma parcial de la expansión en fracciones continuas de x. En la notación de nuestro problema, si el resultado de la medida es y, debemos expandir $y/2^N$. La suma parcial con el denominador más grande menor de 2^{n_0} es el valor j/r que buscamos.

Ejemplo de aplicación. $n_0 = 7$ e $y = 11.490$. ¿Cuál es el valor de j/r? Debemos expandir $\frac{11.490}{2^{14}}$ hasta hallar la suma parcial con el denominador más grande menor que $2^7 = 128$. La expansión en fracciones continuas es:

$$\frac{11.490}{2^{14}} = \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{7 + \cfrac{1}{35 + \ldots}}}}}}$$

Si hacemos la suma parcial con el 35 incluido, obtenemos un denominador mayor que 128. Sin el 35, obtenemos $\frac{54}{77}$, que es nuestro resultado para j/r.

Notemos que este procedimiento no nos da j y r separadamente, sino la fracción j/r reducida. Es decir, obtenemos j_0 y r_0 sin factores en común y tales que $j_0/r_0 = j/r$. Es un resultado conocido que dados dos números aleatorios, es bastante improbable que tengan factores en común, así que la probabilidad de que j y r los tengan es pequeña. Si los tienen, obtenemos un r_0 que simplemente es un divisor de r. En este caso podemos probar a calcular $b^{kr_0} \bmod n$ para algunos k, y ver si encontramos el periodo. Si esto falla, lo que implicaría que j y r comparten un factor común elevado, repetiríamos la computación cuántica de nuevo, obteniendo la fracción j'/r, que nos proporciona otro divisor de r, r'_0. De nuevo lo más probable es que j y j' no tengan factores en común, de modo que r será el mínimo común múltiplo de r_0 y r'_0. En otro caso, repetimos el proceso de comprobar si los múltiplos pequeños de r'_0 son el periodo de la función. Podríamos incluso repetir de nuevo todo el proceso, obteniendo otro múltiplo de $1/r$.

Teniendo en cuenta que en realidad no estamos seguros de que nuestra medida nos haya dado uno de los y_j antes mencionados, para tener éxito deberemos repetir el procedimiento algunas veces, pero no muchas. De hecho, en [5] se prueba que, añadiendo un pequeño número de Qbits al registro de entrada, la probabilidad de obtener r en una sola iteración del algoritmo está muy próxima a uno.

2.6.4. Factorizar números enteros

Como hemos visto, poder encontrar el periodo de la función $f(x) = b^x \bmod n$ supone poder romper mensajes cifrados según el criptosistema RSA. Resulta también interesante ver como un adversario puede usar este algoritmo para factorizar $n = pq$, y por tanto obtener la clave secreta d y romper definitivamente el criptosistema. Seguiremos de nuevo lo expuesto en [5].

Tomamos un número aleatorio a. Las posibilidades de que a sea un múltiplo de p o de q son minúsculas cuando p y q son grandes. En cualquier caso, podemos asegurarnos que esto no ocurre aplicando el algoritmo de Euclides para el cálculo del máximo común divisor de a y n. Si no es 1, habremos obtenido uno de los primos p y q, y ya habríamos factorizado n. En caso contrario, tenemos un a coprimo con n. Usando nuestro algoritmo para encontrar periodos, encontramos el orden de a en G_{pq}, esto es: el r más pequeño para el que $a^r \equiv 1 \bmod pq$. Si tenemos suerte, esta información será suficiente para calcular la factorización de n.

Supongamos en primer lugar que hemos obtenido un r par. Podemos calcular entonces $x \equiv a^{r/2} \bmod pq$. Este x verifica

$$0 \equiv x^2 - 1 \bmod pq \equiv (x-1)(x+1) \bmod pq. \tag{2.4}$$

Notar que $x - 1 = a^{r/2} - 1$ no es congruente con 0 módulo pq, pues r es el mínimo exponente para el que a^r es congruente con 1 módulo pq. Hagamos ahora la suposición de que también se verifica que

$$x + 1 = a^{r/2} + 1 \not\equiv 0 \bmod pq.$$

En este caso ni $x-1$ ni $x+1$ son divisibles por $n = pq$, pero en virtud de (2.4) su producto sí. Como p y q son ambos primos, esto significa que uno de ellos, por ejemplo p divide a $x-1$ y el otro, digamos q, divide a $x+1$. Como los únicos divisores de n son p y q, se sigue que $mcd(n, x-1) = p$ y $mcd(n, x+1) = q$. Usando el algoritmo euclídeo, obtendríamos los primos p y q.

Hace falta tener un poco de suerte para elegir un a tal que su orden r sea par y verifique $a^{r/2} + 1 \not\equiv 0 \bmod pq$, de modo que podamos obtener la factorización de n. En [5] se prueba que la probabilidad de obtener tal a es al menos 0,5. De este modo, no necesitaremos repetir el proceso muchas veces para tener éxito. Resulta por tanto sencillo factorizar n con un ordenador cuántico, quedando expuesta la clave secreta del criptosistema.

2.7. El algoritmo de Shor para el problema del logaritmo discreto

Hemos visto que con un ordenador cuántico toda la criptografía basada en la suposición RSA quedará obsoleta. ¿Pasará lo mismo con la criptografía basada en el logaritmo discreto? La respuesta es que sí: una variante del algoritmo de Shor permite resolver el problema del logaritmo discreto de forma eficiente. A continuación se da una descripción de este algoritmo basada en [6].

Como en el primer capítulo, partimos de un grupo cíclico $\mathbb{G} =< g >$ de orden q. Dado un elemento $x \in \mathbb{G}$, queremos calcular $\log_g x$, el menor entero α tal que $g^\alpha = x$. Como hemos visto, el elemento α se puede tomar en el grupo aditivo $\mathbb{Z}_q$. Suponemos que $x \neq g$ (en caso contrario ya estaría resuelto el problema del logaritmo discreto) y definimos

$$f : (\alpha, \beta) \in \mathbb{Z}_q \times \mathbb{Z}_q \longrightarrow f(\alpha, \beta) = x^\alpha g^{-\beta} \in \mathbb{G}.$$

Como $f(\alpha,\beta) = g^{\alpha \log_g x - \beta}$, f es constante en

$$\{(\alpha,\beta) \in \mathbb{Z}_q^2 : \alpha \log_g x - \beta = cte\};$$

es decir:

$$f(\alpha,\beta) = f(\alpha',\beta') \Longleftrightarrow (\alpha-\alpha',\beta-\beta') = (k, k\log_g x), \text{ para } k = 0,\ldots,q-1.$$

En términos del grupo aditivo $\mathbb{Z}_q \times \mathbb{Z}_q$:

$$((\alpha-\alpha',\beta-\beta') \in H = \{(0,0),(1,\log_g x),\ldots,(q-1,(q-1)\log_q x)\}.$$

Recordando resultados de álgebra, esto quiere decir que $f(\alpha,\beta) = f(\alpha',\beta') \Longleftrightarrow (\alpha',\beta') \in (\alpha,\beta)+H$, esto es, la clase de (α,β) módulo H (como $\mathbb{Z}_q \times \mathbb{Z}_q$ es abeliano no tenemos que distinguir clases a izquierda y clases a derecha). En lo siguiente, serán de gran importancia las clases módulo H de la forma:

$$(0,\delta)+H = \{(\alpha,\delta+\alpha\log_g x) : \alpha \in \mathbb{Z}_q\}.$$

El algoritmo procede de la siguiente manera. Nuestro registro de entrada estará compuesto por dos registros, que pondremos en el estado

$$|\Psi\rangle = \frac{1}{q}\sum_{\alpha,\beta\in\mathbb{Z}_q} |\alpha,\beta\rangle\,. \tag{2.5}$$

Para llegar a este estado, se utilizan en cada registro $n+1$ Qbits, siendo n el número de bits en la representación binaria de q. Utilizando la transformación $\mathbf{H}^{\otimes n}$, se ponen los n primeros Qbits en el estado combinación lineal con todos los coeficientes iguales, de modo que los $n+1$ Qbits quedan en el estado

$$\frac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n-1} |x\rangle_n\,|0\rangle\,.$$

A continuación se define la función $g(x)$ tal que $g(x) = 1$ si $x = 0,\ldots,q-1$ y $g(x) = 0$ en otro caso. Se aplica la transformación $\mathbf{U}_g$ a nuestro estado y obtenemos

$$\frac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n-1} |x\rangle_n\,|g(x)\rangle\,.$$

Medimos el último Qbit de este estado. Si obtenemos un 1, aplicando la regla de Born generalizada sabemos que los n primeros Qbits habrán quedado en el estado

$$\frac{1}{\sqrt{q}}\sum_{x=0}^{q-1} |x\rangle\,. \tag{2.6}$$

Como n es el número de bits en la representación binaria de q, tenemos que $2^{n-1} \leq q < 2^n$, de modo que tenemos al menos una probabilidad $1/2$ de haber medido un $|1\rangle$ y haber dejado el sistema en el estado (2.6). Si hemos medido un $|0\rangle$, repetimos todo el proceso, y es muy probable que en pocos intentos obtengamos un $|1\rangle$.

Así, aplicando este proceso a ambos registros de $n+1$ Qbits, hemos dejado cada uno en el estado (2.6). Haciendo ahora el producto tensorial, obtenemos el estado deseado en la forma de (2.5).

Ahora, utilizando un número de Qbits suficientemente grande en el registro de salida (lo suficiente para representar de algún modo elementos de $\mathbb{G}$), aplicamos la transformación $\mathbf{U}_f$ para obtener el estado

$$\frac{1}{q} \sum_{\alpha,\beta \in \mathbb{Z}_q} |\alpha, \beta, f(\alpha, \beta)\rangle \, .$$

Medimos el registro de salida. Si obtenemos $g^{-\delta}$, el registro de entrada se habrá quedado en un estado correspondiente a la clase $(0, \delta) + H$, esto es en el estado

$$\frac{1}{\sqrt{q}} \sum_{\alpha \in \mathbb{Z}_q} \left|\alpha, \delta + \alpha \log_g x\right\rangle .$$

En principio no podemos obtener este δ, pues obtenerlo equivaldría a resolver el problema del logaritmo discreto. Lo que sí que podemos hacer es aplicar la transformada cuántica de Fourier sobre $\mathbb{Z}_q$ a cada uno de los dos registros. En general, para $N \in \mathbb{N}$, si $x \in \mathbb{Z}_q$, esta transformación viene dada por:

$$\mathbf{U}_{FT} \, |x\rangle = \frac{1}{\sqrt{N}} \sum_{y \in \mathbb{Z}_N} e^{2\pi i x y / N} \, |y\rangle \, .$$

Esta transformación es el caso más general de la transformación dada en la sección anterior, donde N era una potencia de 2. Aquel caso es más sencillo de implementar en un ordenador cuántico, pero esta que usamos ahora también se puede implementar. De hecho, en su paper original de 1994 ([7]), Shor utilizó una transformada cuántica de Fourier sobre $\mathbb{Z}_N$, con N un número en cuya factorización solo aparecen primos 'pequeños'.

En cualquier caso, aplicando esta transformación a cada uno de los dos registros, obte-

nemos el estado

$$\frac{1}{q^{3/2}} \sum_{\alpha \in \mathbb{Z}_q} \left(\sum_{\mu \in \mathbb{Z}_q} e^{2\pi i \mu \alpha / q} \left| \mu \right\rangle \right) \left(\sum_{\nu \in \mathbb{Z}_q} e^{2\pi i \nu (\delta + \alpha \log_g x)/q} \left| \nu \right\rangle \right) =$$
$$= \frac{1}{q^{3/2}} \sum_{\alpha, \mu, \nu \in \mathbb{Z}_q} e^{2\pi i [\mu \alpha + \nu (\delta + \alpha \log_g x)]/q} \left| \mu, \nu \right\rangle =$$
$$= \frac{1}{q^{3/2}} \sum_{\mu, \nu \in \mathbb{Z}_q} e^{2\pi i \nu \delta / q} \sum_{\alpha \in \mathbb{Z}_q} e^{2\pi i \alpha (\mu + \nu \log_g x)/q} \left| \mu, \nu \right\rangle .$$

Usando la identidad $\sum_{\alpha \in \mathbb{Z}_q} e^{2\pi i \alpha \beta / q} = q \delta_{\beta, 0}$, tenemos el estado

$$\frac{1}{\sqrt{q}} \sum_{\nu \in \mathbb{Z}_q} e^{2\pi i \nu \delta / q} \left| -\nu \log_g x, \nu \right\rangle .$$

Por último, hacemos una medida de este estado, obteniendo un par $(-\nu \log_g x, \nu)$ para algún $\nu \in \mathbb{Z}_q$. Si ν tiene inverso multiplicativo módulo q, dividimos por $-\nu$ la primera componente y obtenemos el logaritmo deseado. Se puede probar que la probabilidad de tener éxito en este proceso es bastante elevada, de modo que no tenemos que repetir el proceso tantas veces obtener la respuesta deseada.

Resumen del capítulo

En este capítulo hemos visto una introducción al proceso computacional de los ordenadores cuánticos, así como una descripción de los algoritmos cuánticos que resuelven eficientemente los problemas de factorizar enteros y calcular logaritmos discretos. En el conjunto del TFG, el capítulo ha servido para comprender cómo funcionan los algoritmos cuánticos que rompen las técnicas criptográficas usadas actualmente, haciendo necesaria la criptografía postcuántica. También, han quedado claras las diferencias entre ordenador clásico y ordenador cuántico que hacen que los algoritmos vistos no se puedan implementar eficientemente en un ordenador clásico.

Capítulo 3

Criptografía postcuántica basada en retículos

En los capítulos anteriores ha quedado patente la necesidad de implementar nuevas técnicas criptográficas que sean resistentes a ataques cuánticos. Hasta la fecha se han desarrollado varias familias de algoritmos criptográficos postcuánticos, entre las que se encuentran la criptografía basada en retículos, la criptografía basada en códigos correctores, o la criptografía basada en funciones polinomiales multivariables. En esta memoria nos centraremos en la criptografía basada en retículos, ya que los esquemas Kyber y Dilithium que buscamos explicar pertenecen a esta familia. Este capítulo constituye una introducción al concepto de retículo, así como a algunas técnicas criptográficas intuitivas basados en ellos, y sigue lo expuesto en las fuentes [8], [9] y [10], con algunos ejemplos tomados de [11].

3.1. ¿Qué son los retículos?

En la siguiente definición, tomaremos los vectores como vectores columna.

Definición 3.1.1. Dada B una matriz $n \times n$ real inversible, un *retículo* generado por B es el conjunto

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

B recibe el nombre de *base* del retículo, y n es su *dimensión*.

Intuitivamente, los retículos son vectores de un espacio de dimensión n que se pueden obtener como combinaciones lineales con coeficientes enteros de los vectores formados por

las columnas de la matriz B. Por ejemplo, un retículo en dimensión 2 aparece en la figura 3.1.

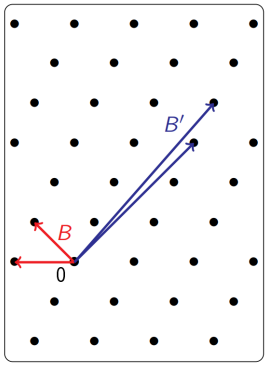

Figura 3.1: Retículo en dimensión dos. Los puntos representan los lugares donde inciden los vectores. Imagen obtenida de [8].

En esta imagen vemos también que la base de un retículo no es única. Este es un hecho muy interesante que nos permitirá desarrollar criptografía basada en retículos, ya que algunos problemas serán complicados de resolver en algunas bases, mientras que serán muy fáciles en otras.

3.2. Problemas de retículos

3.2.1. Problemas de caso peor

Definamos en primer lugar los problemas de retículos intuitivamente más sencillos: los problemas SVP (Shortest Vector Problem), CVP (Closest Vector Problem) y SIVP (Shortest Independent Vector Problem). Éstos son problemas de caso peor, lo que quiere decir que no existe un algoritmo eficiente que resuelva cualquier instancia, pero puede ser que alguna instancia del problema sea fácil de resolver. Estos problemas, que se basan en el uso de la norma euclídea en $\mathbb{R}^n$, se definen como sigue:

- SVP: dada una base B, encontrar el vector más corto distinto de cero en $\mathcal{L}(B)$.
- CVP: dada una base B y un vector t (no necesariamente en el retículo), encontrar el elemento del retículo $v \in \mathcal{L}(B)$ más cercano a t.
- SIVP: dada una base B, encontrar n vectores linealmente independientes $\mathbf{S} = [s_1, \ldots, s_n]$, con $s_i \in \mathcal{L}(B)$ tales que la cantidad $\|\mathbf{S}\| = \max \|s_i\|$ sea mínima.

También se pueden considerar variantes aproximadas de estos problemas, que se indican con el prefijo $\gamma-$, donde γ es el factor de aproximación. Por ejemplo, el problema $\gamma-$SVP consiste en encontrar un vector cuya norma sea como máximo γ veces la del vector más corto.

Observación. Todos estos problemas se pueden definir con cualquier norma, pero como hemos dicho en general usamos la norma euclídea.

Estos problemas son difíciles de resolver para todas las instancias, incluso con un ordenador cuántico, cuando n es grande. Esto quiere decir que no existe un algoritmo que los resuelva en tiempo polinomial. Si consideramos los problemas aproximados, serán más fáciles de resolver cuanto más grande sea el factor de aproximación. La figura 3.2 muestra, para los problemas γ-SVP y γ-CVP una gráfica del tiempo que se tarda en resolver un problema aproximado con el algoritmo más eficiente conocido, según va creciendo el factor de aproximación.

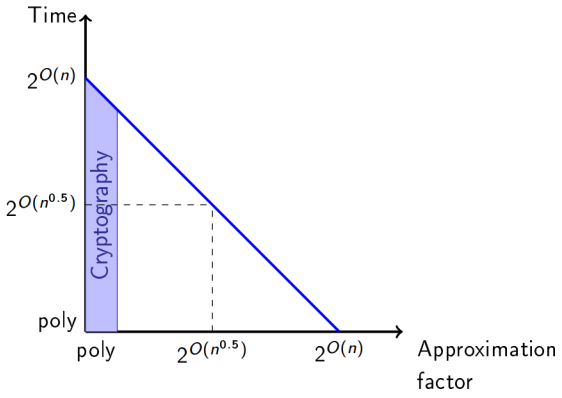

Figura 3.2: Tiempo de resolución vs. factor de aproximación. Imagen obtenida de [8]

En la figura se muestran también los problemas que son susceptibles de aplicación criptográfica. Como vemos, para que el problema sea difícil necesitamos un factor de aproximación pequeño.

3.2.2. Problemas de caso medio

Como hemos visto, los problemas tratados en la sección anterior son problemas de caso peor, lo que significa que pueden existir instancias del problema que sean fáciles de resolver. Por tanto, no son los mejores problemas sobre los que desarrollar criptografía, sino que

debemos desarrollar problemas que sean difíciles en el caso medio. Esto quiere decir que dada una instancia aleatoria del problema, el problema será difícil de resolver con casi toda probabilidad. En esta sección describiremos alguno de estos problemas.

Definición 3.2.1 (Problema SIS). Sean q y B números enteros con $1 \leq B \ll q$. Dada una matriz $A \in \mathbb{Z}_q^{m \times n}$, con $n \log q < m$, elegida de manera aleatoria y uniforme, el problema SIS (Short Integer Solution) consiste en encontrar $x \in \mathbb{Z}^m$ tal que $x^t A \equiv 0 \bmod q$, con $x \neq 0$ y $\|x\| \leq B$.

Este problema se puede ver como un problema de retículos. En efecto, definiendo el retículo $L := \{x \in \mathbb{Z}^m / x^t A \equiv 0 \bmod q\}$, el problema SIS equivale a resolver un problema SVP aproximado en L, que es un retículo de dimensión m.

Además, se ha probado que resolver el problema SIS con probabilidad no nula es más difícil que resolver el problema SIVP aproximado en *cualquier* retículo de dimensión n. Esto viene dado por el siguiente teorema, dado en [8]:

Teorema 3.2.1. Para cualquier $m = poly(n)$, $B > 0$ y $q \geq B \cdot poly(n)$ suficientemente grande, un problema γ-SIVP en un retículo arbitrario de dimensión n, con $\gamma = B \cdot poly(n)$, se reduce a un problema SIS.

Esto significa que resolver el problema SIS implica resolver un problema γ-SIVP arbitrario, por lo que resolver el problema SIS debe ser más difícil que resolver *cualquier* problema γ-SIVP.

Definición 3.2.2 (Problema LWE). Sean q y B números enteros con $1 \leq B \ll q$. Sea una matriz $A \in \mathbb{Z}_q^{m \times n}$, elegida de manera aleatoria y uniforme. Por otro lado, dada una distribución de probabilidad χ_B sobre $\{-B, \ldots, B\}$, tomamos vectores $s \in \mathbb{Z}^n$ y $e \in \mathbb{Z}^m$, cuyos coeficientes son tomados aleatoriamente según χ_B (de ahora en adelante denotaremos esto como $s \leftarrow \chi_B^n$ y $e \leftarrow \chi_B^m$). Dados A y $b := As + e \bmod q$, el problema LWE (Learning With Errors), consiste en obtener los vectores s y e.

Observación. En ocasiones la distribución de probabilidad se toma sobre $\mathbb{Z}_q$, lo que da lugar a un desarrollo prácticamente equivalente.

De nuevo, podemos ver el problema LWE como un problema de retículos. Para $n = m$, definimos el retículo $L = \{x \in \mathbb{Z}^n / \exists s \in \mathbb{Z}^n \text{ con } As \equiv x \bmod q\}$. Tal y como aparece en la figura 3.3, se nos daría el retículo L y el punto b, y tendríamos que ser capaces de obtener el punto v, es decir, resolver un problema CVP en L.

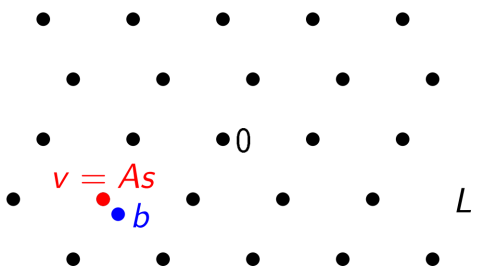

Figura 3.3: Imagen tomada de [8]

Como para el problema SIS, se han hecho reducciones de problemas SVP y SIVP aproximados al problema LWE. En un primer momento, esta reducción utilizaba un algoritmo cuántico, de modo que resolver LWE implicaba un algoritmo cuántico para resolver los problemas SVP y SIVP aproximados. Esto era algo más débil que una reducción 'clásica', pues obtener un algoritmo cuántico para resolver problemas de retículos parece más factible que obtener un algoritmo clásico. Sin embargo, recientemente se ha conseguido hacer esta reducción clásicamente, por lo que al final esta reducción resulta ser tan fuerte como la que teníamos para el problema SIS.

Una variante muy útil para criptografía es la del problema LWE de decisión, que se define a continuación:

Definición 3.2.3 (Problema LWE de decisión). Sea χ_B una distribución de probabilidad sobre $\{-B, \ldots, B\}$, y sean $s \leftarrow \chi_B^n$ y $e \leftarrow \chi_B^m$. Tomamos aleatoria y uniformemente una matriz $A \in \mathbb{Z}_q^{m \times n}$. Dados la matriz A y $b \in \mathbb{Z}_q^m$, donde b puede ser un vector aleatorio elegido uniformemente o puede calcularse según $b = As + e \mod q$, decidir si b es el aleatorio o no.

Este problema de decisión es equivalente en dificultad al problema LWE habitual. De igual modo, se puede probar ([8]) que los problemas SIS y LWE son equivalentes cuánticamente; es decir: existen algoritmos cuánticos que reducen un problema al otro.

3.3. Algunas técnicas criptográficas basadas en problemas de retículos

En esta sección veremos criptosistemas y esquemas de firma digital intuitivos y basados en los problemas de retículos.

3.3.1. El criptosistema de clave pública GGH

Este criptosistema de clave pública fue propuesto por Goldreich, Goldwasser y Halevi en 1997. En la práctica no es recomendable usar este criptosistema, pues se han encontrado ataques efectivos contra él. Aun así, es interesante considerarlo como primer ejemplo, debido a su simplicidad. En [11], su funcionamiento se describe de la siguiente manera:

- La clave privada es una base B 'buena' para un retículo. Típicamente esta base estará formada por vectores cortos y ortogonales entre sí, lo que nos puede facilitar resolver problemas como el CVP.
- La clave pública es una base 'mala' H para el mismo retículo, típicamente vectores largos y no ortogonales.
- El proceso de cifrado consiste en tomar un elemento del retículo v y añadirle cierto ruido r. El mensaje cifrado sería $c = v + r$.
- El proceso de descifrado consiste en, dado $c = v + r$, encontrar el vector del retículo v. Si el ruido r es suficientemente pequeño, esto consiste en un problema CVP.

Algunos inconvenientes de este criptosistema son que no es semánticamente seguro, pues el proceso de cifrado no es aleatorio, o que para claves de tamaño moderado se han desarrollado ataques que rompen el criptosistema. Esto se podría solucionar aumentando el tamaño de los parámetros del sistema, pero al final resultaría ineficiente trabajar con tales parámetros.

3.3.2. El criptosistema de clave pública basado en el problema LWE

Este criptosistema fue desarrollado en primer lugar por [12] y durante algún tiempo fue el criptosistema basado en retículos más eficiente de cuantos disponían de una prueba de seguridad. Posteriormente, se han publicado mejoras en su eficiencia pero, por simplicidad, en este TFG me limitaré a describirlo según aparece en ese primer artículo. El criptosistema solo cifra bits, de modo que el mensaje será 0 o 1. Sus parámetros son los enteros positivos n, llamado parámetro de seguridad, m y p, así como una distribución de probabilidad χ sobre $\mathbb{Z}_p$. Como veremos, una elección de parámetros que garantiza tanto la corrección como la seguridad del criptosistema consiste en tomar $p \geq 2$ primo entre n^2 y $2n^2$, y $m = (1 + \varepsilon)(n + 1) \cdot \log p$, para una constante arbitraria ε.

Para desarrollar el criptosistema, nos harán falta algunas aclaraciones en la notación, que también serán necesarias en el capítulo 4:

- Dado $x \in \mathbb{R}$, definimos $\lfloor x \rfloor$ como el mayor entero menor que x (redondear hacia abajo).
- Dado $x \in \mathbb{R}$, definimos $\lfloor x \rceil$ como el entero más próximo a x, redondeando los empates hacia abajo.
- Para dos reales x e $y > 0$, definimos $x \bmod y$ como $x - \lfloor x/y \rfloor y$.

También nos harán falta dos definiciones:

Definición 3.3.1. Dado $\beta > 0$, la distribución Ψ_β es la distribución en $\mathbb{T} = [0, 1)$ obtenida muestreando una variable aleatoria normal $\mathcal{N}(0, \beta/\sqrt{2\pi})$ y reduciendo el resultado módulo 1.

Definición 3.3.2. Dada una distribución de probabilidad sobre $\mathbb{T}$ con función de densidad $\phi : \mathbb{T} \to \mathbb{R}^+$ y un entero $p \geq 1$, definimos la discretización de ϕ, $\bar{\phi} : \mathbb{Z}_p \to \mathbb{R}^+$, como la distribución de probabilidad obtenida muestreando de ϕ, multiplicando por p y redondeando al entero más cercano módulo p. En particular, la variable $\bar{\Psi}_\beta$ consiste en muestrear una normal $\mathcal{N}(0, \beta p/\sqrt{2\pi})$ y redondear al entero más cercano módulo p.

En el criptosistema, la distribución de probabilidad χ sobre $\mathbb{Z}_p$ que se usa es $\bar{\Psi}_{\alpha(n)}$, con $\alpha(n) = 1/(\sqrt{n} \log^2 n)$.

Ahora estamos en condiciones de describir los algoritmos de generación de claves, cifrado y descifrado:

- $G(\cdot) :=$
  1. Tomar $a_1, \ldots, a_m \in \mathbb{Z}_p^n$, de manera aleatoria y uniforme.
  2. Tomar $e_1, \ldots, e_m \in \mathbb{Z}_p$,, independientemente y según la distribución χ.
  3. Tomar $s \in \mathbb{Z}_p^n$, de manera aleatoria y uniforme.
  4. Devolver $pk = (a_i, b_i)_{i=1}^m$, donde $b_i = a_i^T \cdot s + e_i \bmod p$, y $sk = s$. T denota la trasposición del vector.
- Dado un bit $t \in \{0, 1\}$, para calcular $E(pk, t)$:
  1. Tomar de manera aleatoria y uniforme un subconjunto S, entre los 2^m subconjuntos de $\{1, \ldots, m\}$.

2. Devolver $c = (a, b) = \left(\sum_{i \in S} a_i, t \cdot \lfloor p/2 \rfloor + \sum_{i \in S} b_i\right)$, donde todas las sumas se hacen módulo p.

- Dado $c = (a, b)$, para calcular $D(sk, c)$:

1. Calcular $x = b - a^T \cdot s \bmod p$.
2. Devolver 0 si x está más cerca de 0 que de $\lfloor p/2 \rfloor \bmod p$, y 1 en otro caso.

A diferencia de los esquemas de cifrado del capítulo 1, este esquema no siempre descifrará de forma correcta. A continuación se ven dos definiciones y dos resultados que nos dan su $(1 - \delta)-$corrección. Ambos resultados están probados en [12].

Definición 3.3.3. Sea χ una distribución de probabilidad sobre $\mathbb{Z}_p$, y sea $k \geq 0$ un entero. Definimos la distribución χ^{*k} como la distribución obtenida de sumar en $\mathbb{Z}_p$ k muestras independientes de χ. Para $k = 0$, se define χ^{*0} como la distribución degenerada en 0.

Definición 3.3.4. Dado un elemento $a \in \mathbb{Z}_p$, definimos $|a|$ como el entero a si $a \in \{0, \ldots, \lfloor p/2 \rfloor\}$, y el entero $p - a$ en otro caso. $|a|$ representa 'la distancia de a a 0'. De manera similar, para $x \in \mathbb{T}$, definimos $|x|$ como x si $x \in [0, 1/2]$ y como $1 - x$ en otro caso.

Lema. Sea $\delta > 0$. Supongamos que, para todo $k \in \{0, \ldots, m\}$, χ^{*k} satisface que

$$P\left(\left|\chi^{*k}\right| < \lfloor p/2 \rfloor / 2\right) > 1 - \delta$$

Entonces, la probabilidad de error de descifrado es como mucho δ. Esto es, para cualquier bit $t \in \{0, 1\}$, y cualquier salida del algoritmo de generación de claves $(pk, sk) \leftarrow G(\cdot)$, se tiene $P(D(sk, E(pk, t)) = t) \geq 1 - \delta$.

Lema. Para nuestra elección de parámetros, se verifica que para todo $k \in \{0, \ldots, m\}$,

$$P\left(\left|\bar{\Psi}^{*k}_{\alpha(n)}\right| < \lfloor p/2 \rfloor / 2\right) > 1 - \delta(n)$$

para una función despreciable $\delta(n)$[1].

Por su parte, el siguiente resultado, también probado en [12], garantiza la seguridad semántica del criptosistema:

Proposición 3.3.1 (Seguridad semántica). Para nuestra elección de los parámetros, si existe un algoritmo que en tiempo polinomial distingue entre los cifrados de 0 y 1, entonces existe un algoritmo que resuelve la mayoría de las instancias del problema LWE de decisión.

[1] $\delta(n)$ es una función despreciable en n si $\lim_{n \to \infty} \delta(n) n^c = 0$ para toda constante $c > 0$

Como hemos visto en el primer capítulo del trabajo, esto implica la CPA seguridad del sistema. El criptosistema propuesto en [12] no es CCA seguro, aunque podríamos convertirlo en CCA seguro usando algún procedimiento que convierte criptosistemas CPA seguros en CCA seguros, como la transformación de Fujisaki-Okamoto, mencionada con anterioridad.

3.3.3. Un esquema de firma digital

Utilizando una idea similar a la del criptosistema GGH se puede desarrollar un esquema de firma digital intuitivo. Consideremos un retículo $\mathcal{L}$ y una función resumen del espacio de mensajes $\mathcal{M}$ a $\mathbb{R}^n$, con n la dimensión del retículo: $H : \mathcal{M} \to \mathbb{R}^n$. Notar que estas funciones resumen no son las pertenecientes al estándar SHA, que siempre devolvían cadenas de ceros y unos. El esquema de firma digital es el siguiente:

- El algoritmo de generación de claves $G(\cdot)$ nos da pk, una base mala del retículo, y sk una base corta y buena, con la que se puede resolver el CVP fácilmente.
- El algoritmo de firma toma un mensaje $m \in \mathcal{M}$, calcula $x = H(m)$ y devuelve $S(sk, m) = s$, donde $s \in \mathcal{L}$ es un elemento del retículo cercano a x.
- El algoritmo de verificación $V(pk, s)$ toma s, comprueba que $s \in \mathcal{L}$ y que $H(m) - s$ es pequeño.

Este esquema de firma digital puede ser sometido a un ataque de mensaje conocido que revela la clave secreta, es decir: rompe totalmente el esquema de firma digital. Este ataque se conoce como ataque del paralelepípedo y funciona de la siguiente manera:

- El adversario conoce unos pares mensaje-firma $(m_1, s_1), (m_2, s_2), \ldots$.
- Dibuja $H(m_i) - s_i$ para cada i.

Como vemos en la figura 3.4, si el número de pares mensaje-firma conocidos es suficientemente grande, el adversario puede encontrar la clave privada (la base buena) a partir de la forma del paralelepípedo.

Una manera de solucionar esto es aleatorizar el algoritmo de firma. Dado un mensaje m, tomamos una distribución normal sobre el retículo (es decir, una normal multidimensional discretizada) centrada en $H(m)$. Tomamos la firma s como una muestra de esta variable aleatoria. Como antes, s está en el retículo y por las propiedades de la normal $H(m) - s$

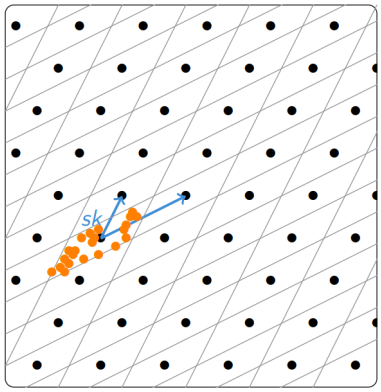

Figura 3.4: Ataque del paralelepípedo. Imagen tomada de [9]

es pequeño. La distribución de $H(m) - s$ es independiente de la clave secreta, así que un adversario no podrá aplicar el ataque del paralelepípedo. De hecho, un resultado en [9], basado en la dificultad de resolver el problema SIS, garantiza que este esquema de firma digital es seguro (existencialmente infalsificable bajo ataque de mensaje escogido).

3.4. Retículos con estructura algebraica

Los problemas de la sección anterior se han definido sobre retículos con bases reales. Se pueden definir también retículos que tienen, además, estructura algebraica; es decir, se construyen sobre anillos o módulos sobre un anillo. Esto nos permitirá hacer operaciones con más facilidad, lo que supondrá una ventaja a la hora de implementar los diferentes criptosistemas que se pueden desarrollar. Sin embargo, la estructura algebraica puede facilitar algunos ataques, así que deberemos buscar un compromiso entre seguridad y eficiencia.

3.4.1. Construcción de los retículos estructurados

Construyamos retículos con estructura basada en el anillo $R = \mathbb{Z}[X]/(p)$, con p polinomio mónico e irreducible de grado d. Para ello, es útil definir la siguiente incrustación del anillo en $\mathbb{C}$:

Definición 3.4.1. Sean $\alpha_1, \ldots, \alpha_d$ las raíces complejas de p. Se define la incrustación canónica como:

$$\sigma : y(X) \in R \longrightarrow (y(\alpha_1), \ldots, y(\alpha_d)) \in \mathbb{C}^d.$$

En estas condiciones, se dice que R es un retículo de dimensión d pues:

$$R = 1 \cdot \mathbb{Z} + X \cdot \mathbb{Z} + \cdots + X^{d-1} \cdot \mathbb{Z}$$

Para hacer una analogía con los retículos vistos anteriormente, se puede probar ([10]) que $\sigma(R)$ es un retículo de dimensión d en $\mathbb{C}^d \cong \mathbb{R}^{2d}$ con base $(\sigma(X^i))_{0 \leq i < d}$, pues

$$\sigma(R) = \sigma(1) \cdot \mathbb{Z} + \sigma(X) \cdot \mathbb{Z} + \cdots + \sigma(X^{d-1}) \cdot \mathbb{Z}$$

Algo análogo se puede decir para un ideal principal (g):

$$(g) = g \cdot R = g \cdot 1 \cdot \mathbb{Z} + \cdots + g \cdot X^{d-1} \cdot \mathbb{Z}$$
$$\sigma((g)) = \sigma(g \cdot R) = \sigma(g) \cdot \mathbb{Z} + \cdots + \sigma(g \cdot X^{d-1}) \cdot \mathbb{Z}$$

Esto es, $\sigma((g))$ es un retículo de dimensión d en $\mathbb{C}^d \approx \mathbb{R}^{2d}$ con base $(\sigma(g \cdot X^i))_{0 \leq i < d}$. Hemos conseguido por tanto obtener retículos a partir de anillos e ideales.

Por su parte, para definir los retículos sobre módulos, hagamos la siguiente observación:

Observación. Sea un anillo conmutativo y con identidad R y B una matriz $k \times k$ de elementos de R con $det(B) \neq 0$. Entonces $M = \{Bx / x \in R^k\}$ es un R-módulo libre cuya base son las columnas de B. En estas circunstancias k recibe el nombre de rango del módulo.

Esto quiere decir que los elementos de M se pueden ver como vectores columna de k elementos del anillo R. De este modo, si

$$m = \begin{pmatrix} m_1 \\ \vdots \\ m_k \end{pmatrix} \in M,$$

se puede aplicar σ componente a componente y escribir

$$\sigma(m) = \begin{pmatrix} \sigma(m_1) \\ \vdots \\ \sigma(m_k) \end{pmatrix}.$$

Con esta notación, en [10] se prueba que $\sigma(M)$ es un retículo de dimensión $n := d \cdot k$ en $\mathbb{C}^n$ con base $(\sigma(b_i X^j))_{1 \leq i \leq k,\ 0 \leq j < d}$, donde b_i, con $1 \leq i \leq k$ son las columnas de B. Tenemos por tanto un retículo construido a partir de un módulo.

3.4.2. Problemas sobre retículos con estructura algebraica

En general, dado un problema X sobre retículos se denota id-X al problema X restringido a retículos sobre ideales, y se denota mod-X_k al problema X restringido a retículos sobre módulos de rango k. Como muestra la figura 3.5, el problema SVP para módulos de rango $k \geq 2$ es tan difícil de resolver como el problema SVP para retículos cualesquiera. Sin embargo, en el problema SVP sobre ideales, esto es en el problema id-SVP, se puede utilizar la estructura algebraica para obtener algoritmos de resolución notablemente más eficientes, especialmente cuando se utilizan algoritmos cuánticos.

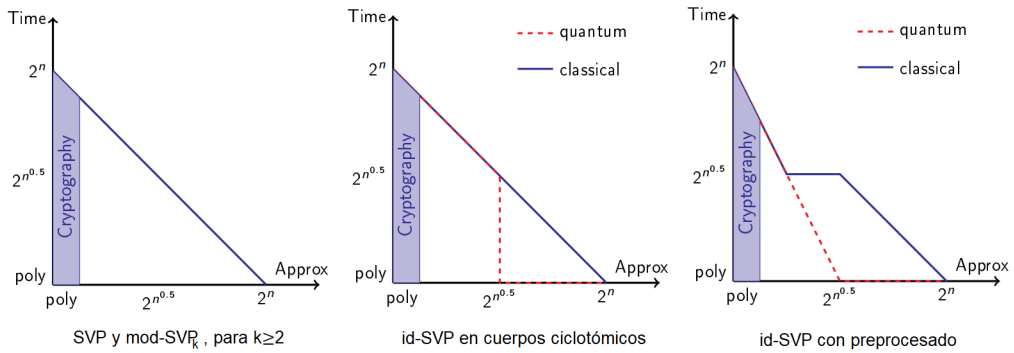

Figura 3.5: Complejidad para algunos problemas sobre retículos con estructura. Imagen tomada de [10]

Debemos tener en cuenta que esto no es una prueba de que el problema mod-SVP_k sea tan seguro como el problema SVP sobre retículos arbitrarios, simplemente significa que hasta la fecha no se han encontrado algoritmos que exploten la estructura de los retículos sobre módulos para crear ataques más eficientes. Cuando avance más el criptoanálisis de los problemas sobre retículos, tal vez aparezcan tales algoritmos.

Un problema muy utilizado en criptografía es el mod-LWE_k, también denotado MLWE.

Definición 3.4.2 (Problema mod-LWE_k). Sean $k, m \in \mathbb{Z}_{>0}$ y χ una distribución de probabilidad sobre R. Dados $(A, b) \in R^{m\times k} \times R^m$ donde:

- A es aleatorio y uniforme en $R_K^{m\times k}$.
- A partir de $s \leftarrow \chi^k$ y $e \leftarrow \chi^m$, se calcula $b = As + e$.

El problema mod-LWE_k consiste en hallar s.

Si $k = 1$, el problema recibe el nombre de problema LWE sobre anillos, y se denota RLWE (Ring LWE).

Como en el caso general, podemos plantear una variante de decisión de este problema, el problema dec-mod-LWE$_k$:

Definición 3.4.3 (Problema dec-mod-LWE$_k$). Sean $k, m \in \mathbb{Z}_{>0}$ y χ una distribución de probabilidad sobre R. El problema dec-mod-LWE$_k$ consiste en distinguir entre (A, b) y (A, u), donde A y b se definen como antes y u es uniforme en R^m.

Como antes, el problema de decisión y el normal (también llamado de búsqueda) resultan ser equivalentes en dificultad. Sin embargo, el problema de decisión será más útil para desarrollar criptografía.

Otro problema famoso sobre módulos con estructura es el problema NTRU. Este problema es un problema basado en anillos que se ha usado para construir un criptosistema, llamado criptosistema NTRU. Este criptosistema es uno de los más eficientes conocidos hasta la fecha. Sin embargo, no está respaldado por una prueba de seguridad que pruebe que romper el criptosistema es al menos tan duro como resolver algún problema de retículos. De hecho, ya han aparecido ataques a este criptosistema que se aprovechan de la estructura algebraica.

Resumen del capítulo

Hemos visto en este capítulo una introducción general al concepto de retículo y a los problemas que se pueden plantear sobre ellos. Se han introducido algunos esquemas intuitivos de cifrado de clave pública y de firma digital, que como hemos visto planteaban algunos problemas en cuanto a seguridad o a eficiencia. Con el fin de mejorar la eficiencia, hemos definido los retículos con estructura algebraica, que nos permitirán desarrollar esquemas mucho más eficientes, aunque deberemos tener cuidado para que la estructura algebraica no facilite ataques. En el conjunto del TFG, este capítulo ha servido para familiarizarnos con nociones de retículos que serán importantes en los últimos dos capítulos, así como para motivar la necesidad de desarrollar esquemas que, como Kyber y Dilithium, sean a la vez seguros y eficientes.

Capítulo 4

Kyber: un KEM CCA seguro

En este capítulo se desarrollará Kyber, la propuesta de CRYSTALS para el concurso del NIST para criptosistemas postcuánticos de clave pública. En líneas generales se seguirá el desarrollo hecho en [13]. Así, en primer lugar, se introduce un criptosistema de clave pública CPA seguro. Después se aplica una variante de la transformación de Fujisaki-Okamoto para obtener un KEM CCA seguro. Por último se desarrolla a partir de este último un criptosistema CCA seguro.

La seguridad de las técnicas criptográficas que se usarán está basada en la dificultad para resolver el problema mod-LWE$_k$. Los esquemas desarrollados son igual de eficientes que los basados en problemas sobre anillos, como el criptosistema NTRU, pero añaden ventajas en flexibilidad y seguridad.

En relación a la flexibilidad, la principal ventaja es que, de normal, cuando uno desarrolla un criptosistema basado en un problema sobre anillos, si quiere modificar los parámetros de seguridad debe cambiar el anillo sobre el que trabaja. De esta forma, para conseguir implementaciones eficientes deberá buscar formas de hacer operaciones eficientemente en el nuevo anillo. En el diseño de Kyber, únicamente se trabaja en el anillo $R_q = \mathbb{Z}_{7681}[X]/(X^{256}+1)$, así que bastará con operar eficientemente en este anillo. Para cambiar los parámetros de seguridad únicamente deberemos cambiar las dimensiones de algunas matrices involucradas.

En cuanto a la seguridad, como ya hemos comentado, se han encontrado ataques usando la estructura algebraica contra los criptosistemas basados en anillos como el NTRU. De los progresos recientes en criptoanálisis, parece extraerse que no será probable que estos ataques se puedan llegar a aplicar al problema mod-LWE.

4.1. Preliminares

4.1.1. ¿Qué es un KEM?

En el desarrollo de Kyber se utiliza el concepto de Key Encapsulation Mechanism (KEM). Éstos son una parte importante de los esquemas híbridos ya vistos en el capítulo 1, en los que mediante un esquema de clave pública ciframos una clave que luego usaremos en un esquema simétrico. El KEM es la parte que se ocupa de cifrar (encapsular) la clave, y a grandes rasgos es algo parecido a un esquema de cifrado de clave pública, pero optimizado para cifrar las claves de un cifrado simétrico. Formalmente, un KEM=(`KeyGen`, `Encaps`, `Decaps`) es una tupla de algoritmos probabilísticos con un espacio de claves $\mathcal{K}$ (de otro criptosistema, por ejemplo uno de clave privada). El algoritmo de generación de claves `KeyGen` devuelve un par de claves (pk, sk). El algoritmo de encapsulamiento hace $(K, c) \leftarrow \texttt{Encaps}(pk)$, donde $K \in \mathcal{K}$ y c se dice encapsulamiento de la clave K. El algoritmo de desencapsulamiento $\texttt{Decaps}(sk, K)$ es un algoritmo determinístico que devuelve una clave $K := \texttt{Decaps}(sk, c) \in \mathcal{K}$ o un símbolo especial $\perp \notin \mathcal{K}$ para indicar que c no es un encapsulamiento válido. Decimos que el KEM es $(1 - \delta)$-correcto si

$$P[\texttt{Decaps}(sk, c) = K : (K, c) \leftarrow \texttt{Encaps}(pk)] \geq 1 - \delta$$

donde la probabilidad se toma sobre $(pk, sk) \leftarrow \texttt{KeyGen}(\cdot)$ y los factores involucrados en la aleatoridad de `Encaps`, que habitualmente reciben el nombre de *coins*.

Para un KEM, se define la noción de IND-CCA seguridad (Indistinguishability under Chosen Ciphertext Attacks), similar a la de CCA seguridad para criptosistemas. Para ello, utilizamos un juego entre adversario y retador:

Juego 4.1.1 (Juego IND-CCA$^{\mathcal{A}}$). Dado un adversario $\mathcal{A}$, definimos dos experimentos. Para $b = 0, 1$, el experimento b se desarrolla como sigue:

- El retador ejecuta $(pk, sk) \leftarrow \texttt{KeyGen}(\cdot)$.
- El retador calcula $(K_0^*, c^*) \leftarrow \texttt{Encaps}(pk)$ y elige una clave $K_1^* \in \mathcal{K}$ al azar.
- El retador envía (K_b^*, c^*) al adversario.
- El adversario devuelve un bit $\hat{b} \in \{0, 1\}$. Al adversario se le permite acceder al algoritmo `Decaps` como un oráculo, es decir, tiene acceso a la función $\texttt{DECAPS}(\cdot) = \texttt{Decaps}(sk, \cdot)$. No se le permite que evalúe `DECAPS` en el mensaje cifrado c^*.

El juego acaba con un 1 si $b = \hat{b}$, y con un 0 en otro caso. Indicamos esto con $IND{-}CCA^{\mathcal{A}} \Rightarrow 1$ o $IND - CCA^{\mathcal{A}} \Rightarrow 0$, respectivamente. Intuitivamente, lo que debe hacer el adversario es distinguir encapsulamientos de claves frente a claves escogidas aleatoriamente.

Definición 4.1.1 (IND-CCA seguridad para KEM). Dado un adversario $\mathcal{A}$, definimos a ventaja de $\mathcal{A}$ en el anterior juego como $\text{Adv}_{KEM}^{IND-CCA}(\mathcal{A}) := \left|P[IND - CCA^{\mathcal{A}} \Rightarrow 1] - \frac{1}{2}\right|$. Esta ventaja se corresponde con la capacidad que tiene el adversario para distinguir encapsulamientos lícitos.

4.1.2. Detalles técnicos

La construcción de Kyber requiere de muchos detalles técnicos que usan los conceptos y notaciones que introducimos a continuación:

Con qué anillos trabajaremos

Usaremos los anillos $R = \mathbb{Z}[X]/(X^n + 1)$ y $R_q = \mathbb{Z}_q[X]/(X^n + 1)$, donde $n = 2^{n'-1}$, de tal modo que $X^n + 1$ es el polinomio ciclotómico de grado $2^{n'}$. En Kyber, los valores de n, n' y q son 256, 9 y 7681, respectivamente. Denotaremos los elementos de los anillos R, R_q, $\mathbb{Z}$ y $\mathbb{Z}_q$ con letras normales, los vectores de elementos de anillos en letra minúscula negrita, y las matrices en letra mayúscula negrita. Para un vector $\mathbf{v}$ y una matriz $\mathbf{A}$, denotaremos $\mathbf{v}^T$ ó $\mathbf{A}^T$ su traspuesto.

Reducciones modulares

Dado un entero positivo par (respectivamente impar) α, definimos $r' = r \bmod^{\pm} \alpha$ como el único elemento r' con $-\frac{\alpha}{2} < r' \leq \frac{\alpha}{2}$ (respectivamente $-\frac{\alpha-1}{2} < r' \leq \frac{\alpha-1}{2}$) con $r' \equiv r \bmod \alpha$. Esto es lo que se conoce como *reducción centrada* módulo α.

Dado un número entero positivo α, definimos $r' = r \bmod^{+} \alpha$ como el único elemento r' en el rango $0 \leq r' < \alpha$ tal que $r' \equiv r \bmod \alpha$.

Cuando la representación exacta no sea importante, escribiremos simplemente $r \bmod \alpha$.

Redondeos

Recordando y ampliando las definiciones dadas en el capítulo anterior, dado un $x \in \mathbb{Q}$, denotamos $\lceil x \rfloor$ el redondeo de x al entero más cercano, con los empates redondeados hacia arriba. Denotamos $\lceil x \rceil$ al redondeo hacia arriba.

Tamaño de los elementos

Para un elemento $w \in \mathbb{Z}_q$, definimos $\|w\|_\infty := \left|w \bmod^{\pm} q\right|$. Para un elemento $w = w_0 + w_1 X + \cdots + w_{n-1} X^{n-1} \in R$ definimos las normas l_∞ y l_2 como:

$$\|w\|_\infty := \max_i \|w_i\|_\infty$$
$$\|w\| := \sqrt{\|w_0\|_\infty^2 + \cdots + \|w_{n-1}\|_\infty^2}$$

De manera similar, para un vector $\mathbf{w} = (w_1, \ldots, w_k) \in R^k$, definimos:

$$\|w\|_\infty := \max_i \|w_i\|_\infty$$
$$\|w\| := \sqrt{\|w_0\|^2 + \cdots + \|w_{n-1}\|^2}$$

Función con salida extendible

Sea `Sam` una función con salida extendible; esto es, una función como las de la familia SHAKE vista en el capítulo 1, que toma una cadena de bits x y proporciona una salida que se puede extender a cualquier longitud deseada. Si queremos que las salidas y de `Sam` estén distribuidas de acuerdo a una distribución S, o uniformemente en un conjunto S, escribimos $y \sim S := \texttt{Sam}(x)$. Este procedimiento será completamente determinístico: dada una cadena de bits x, el output es siempre el mismo y.

Distribuciones de probabilidad

Dado un conjunto A, escribiremos $x \leftarrow A$ para denotar que el elemento x se ha tomado aleatoriamente siguiendo una distribución uniforme entre los elementos de A.

Por otro lado, definimos la distribución binomial centrada B_η, para un entero positivo η de la siguiente manera:

$$\text{Tomar } \{(a_i, b_i)\}_{i=1}^{\eta} \leftarrow (\{0,1\}^2)^\eta.$$
$$\text{Devolver } \sum_{i=1}^{\eta} (a_i - b_i).$$

Si $v \in R = \mathbb{Z}[X]/(X^n + 1)$, escribimos $v \leftarrow \beta_\eta$ para indicar que v se ha generado de una distribución donde cada coeficiente se toma de acuerdo a B_η. Similarmente, un vector de polinomios $\mathbf{v} \in R^k$ puede ser generado según la distribución β_η^k.

Compresión y decompresión

Definimos una función $\texttt{Compress}_q(x, d)$ que toma un elemento $x \in \mathbb{Z}_q$ y devuelve un entero en $\{0, \ldots, 2^d - 1\}$, donde $d < \lceil \log_2 q \rceil$. Definimos también una función $\texttt{Decompress}_q(x, d)$ tal que

$$x' = \texttt{Decompress}_q(\texttt{Compress}_q(x, d), d)$$

sea un elemento cercano a x, en concreto:

$$\left| x' - x \bmod^{\pm} q \right| \leq B_q := \lceil \frac{q}{2^{d+1}} \rfloor.$$

Las funciones que usaremos satisfaciendo estos requisitos son:

$$\texttt{Compress}_q(x, d) = \lceil (2^d/q) \cdot x \rfloor \bmod^{+} 2^d$$
$$\texttt{Decompress}_q(x, d) = \lceil (q/2^d) \cdot x \rfloor$$

Si elegimos $x \in \mathbb{Z}_q$ aleatorio y calculamos $x' = \texttt{Decompress}_q(\texttt{Compress}_q(x, d), d)$, la distribución de $x' - x \bmod^{\pm} q$ es casi uniforme sobre los enteros de magnitud casi B_q. En particular, la distribución tiene la misma probabilidad para los enteros con magnitud menor o igual que $B_q - 1$, y más pequeña para aquellos con magnitud B_q.

A veces usaremos también estas funciones sobre polinomios o vectores de polinomios. En esos casos, las funciones se aplicarán individualmente a cada coeficiente de cada polinomio.

¿Por qué se definen estas funciones? Se hace para descartar bits en la clave pública y en el texto cifrado que no tienen mucho peso en la probabilidad de corrección del proceso de descifrado. De esta forma, podremos utilizar parámetros más pequeños. También, usaremos la función `Compress` en el proceso de descifrado.

4.1.3. El problema MLWE

Como ya se ha mencionado, el problema sobre retículos sobre el que se basa la seguridad de Kyber es el problema mod-LWE, concretamente su variante de decisión. En particular, el problema utilizado es el de la definición 3.4.3, donde el anillo sobre el que se trabaja es R_q y la distribución de probabilidad β_η, para un η dado como parámetro. Para probar los diferentes resultados, será conveniente definir la ventaja de un adversario $\mathcal{A}$ en resolver este problema. Lo haremos una vez más con un juego.

Juego 4.1.2. Dado un adversario $\mathcal{A}$, definimos dos experimentos.

- **Experimento 0:**
  - El retador toma $\mathbf{A} \leftarrow R_q^{m \times k}$ y $(\mathbf{s}, \mathbf{e}) \leftarrow \beta_\eta^k \times \beta_\eta^m$.
  - El retador calcula $\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e}$ y manda $(\mathbf{A}, \mathbf{b})$ al adversario.
  - El adversario devuelve un $\hat{b} \in \{0, 1\}$.
- **Experimento 1:**
  - El retador toma $\mathbf{A} \leftarrow R_q^{m \times k}$ y $\mathbf{b} \leftarrow R_q^m$ y se lo manda al adversario.
  - El retador devuelve un $\hat{b} \in \{0, 1\}$.

Si W_b es el suceso en que el adversario $\mathcal{A}$ saca un 1 en el experimento b, definimos la ventaja de $\mathcal{A}$ en este juego como $\mathrm{Adv}_{m,k,\eta}^{mlwe}(\mathcal{A}) := |P(W_0) - P(W_1)|$.

Definición 4.1.2 (Suposición MLWE). Diremos que la suposición MLWE es válida para (m, k, η) si para todos los adversarios eficientes $\mathcal{A}$, se tiene que $\mathrm{Adv}_{m,k,\eta}^{mlwe}(\mathcal{A})$ es despreciable. Asumir esta suposición representa la idea de que un adversario no es capaz de distinguir vectores aleatorios de vectores próximos a los del retículo.

4.2. El criptosistema CPA seguro

Describiremos en esta sección el primer criptosistema propuesto en [13], y probaremos resultados sobre su corrección y su seguridad.

Sean k, d_t, d_u, d_v, η parámetros enteros positivos. Sea $\mathcal{M} = \{0, 1\}^{256}$ el espacio de los mensajes y notemos que cada mensaje $m \in \mathcal{M}$ se puede expresar como un polinomio en $R = \mathbb{Z}[X]/(X^n + 1)$ con coeficientes en $\{0, 1\}$. El criptosistema Kyber.CPA = (`KeyGen`, `Enc`, `Dec`) se define como sigue:

- El algoritmo de generación de claves $\texttt{KeyGen}(\cdot)$ hace:
  1. $\rho, \sigma \leftarrow \{0, 1\}^{256}$.
  2. $\mathbf{A} \sim R_q^{k \times k} := \texttt{Sam}(\rho)$.
  3. $(\mathbf{s}, \mathbf{e}) \sim \beta_\eta^k \times \beta_\eta^k := \texttt{Sam}(\sigma)$.
  4. $\mathbf{t} := \texttt{Compress}_q(\mathbf{A}\mathbf{s} + \mathbf{e}, d_t)$.
  5. Devolver $(pk := (\mathbf{t}, \rho), sk := \mathbf{s})$.
- El algoritmo de cifrado $\texttt{Enc}(pk = (\mathbf{t}, \rho), m \in \mathcal{M})$ hace:

1. $r \leftarrow \{0,1\}^{256}$.
2. $\mathbf{t} := \texttt{Decompress}_q(\mathbf{t}, d_t)$.
3. $\mathbf{A} \sim R_q^{k \times k} := \texttt{Sam}(\rho)$.
4. $(\mathbf{r}, \mathbf{e}_1, e_2) \sim \beta_\eta^k \times \beta_\eta^k \times \beta_\eta := \texttt{Sam}(r)$.
5. $\mathbf{u} := \texttt{Compress}_q(\mathbf{A}^T\mathbf{r} + \mathbf{e}_1, d_u)$.
6. $v := \texttt{Compress}_q(\mathbf{t}^T\mathbf{r} + e_2 + \lceil \frac{q}{2} \rfloor \cdot m, d_v)$.
7. Devolver $c := (\mathbf{u}, v)$.

Notemos que $c = (\mathbf{u}, v) \in \{0,1\}^{256kd_u} \times \{0,1\}^{256d_v}$.

- El algoritmo de descifrado $\texttt{Dec}(sk = \mathbf{s}, c = (\mathbf{u}, v))$ hace:

1. $\mathbf{u} := \texttt{Decompress}_q(\mathbf{u}, d_u)$.
2. $v := \texttt{Decompress}_q(v, d_v)$.
3. Devolver $\texttt{Compress}_q(v - \mathbf{s}^T\mathbf{u}, 1)$.

De ahora en adelante nos referiremos a estos algoritmos con el prefijo `Kyber.CPA` (por ejemplo `Kyber.CPA.Enc`). Veamos a continuación qué se está haciendo en cada uno de ellos.

El algoritmo `Kyber.CPA.KeyGen` genera dos semillas aleatorias ρ y σ. A partir de estas semillas genera la matriz $\mathbf{A}$, que será la base del módulo que determina el retículo sobre el que trabajamos, y los vectores $\mathbf{s}$ y $\mathbf{e}$. Posteriormente, calcula $\mathbf{t}$ como un vector próximo a un elemento del retículo, al que además se le quitan algunos bits. La clave pública es entonces $(\mathbf{t}, \rho)$ y la privada $\mathbf{s}$. Notemos que lo que se está haciendo es bastante parecido a lo que se hacía en el criptosistema basado en el problema LWE del capítulo 3, con dos diferencias fundamentales (aparte de que el retículo que usamos ahora es estructurado). En primer lugar la matriz $\mathbf{A}$ y el vector $\mathbf{s}$ no se toman según una distribución uniforme, sino según una distribución que depende de la semilla aleatoria σ. Por otro lado, la clave pública es notablemente menos pesada, pues no se da la matriz de todos los elementos, sino una semilla ρ que la determina unívocamente a través la función `Sam`.

El algoritmo `Kyber.CPA.Enc` también hace algo bastante parecido al del criptosistema basado en el problema LWE del capítulo 3. Se toma un r aleatorio, que a través de la función `Sam` determina los vectores $\mathbf{r}$, $\mathbf{e}_1$ y el escalar e_2. Se calcula la matriz $\mathbf{A}$ gracias a la semilla ρ de la clave pública y se procede al encriptado de la cadena de bits m de un modo bastante parecido al del capítulo 3, con la diferencia de que en aquel caso la aleatoriedad la obteníamos del subconjunto S sobre el que sumábamos, y ahora la obtenemos de r. Otra diferencia es que ahora se usa la función `Compress` para despreciar algunos bits.

El algoritmo `Kyber.CPA.Dec` recupera en primer lugar el mensaje cifrado mediante la función `Decompress`. Después la utiliza otra vez para devolver una cadena de 0s y 1s que será el descifrado de $c = (\mathbf{u}, v)$. Para cada coeficiente del polinomio $v - \mathbf{s}^T\mathbf{u}$, como en el criptosistema del capítulo 3, se devuelve un 1 si el coeficiente está más cerca de $\lceil q/2 \rfloor$ que de 0, y un 0 en otro caso.

Ahora veremos un teorema que prueba la $(1-\delta)$-corrección de este criptosistema. En [13] se escogen parámetros de tal forma que $\delta = 2^{-142}$. Vemos antes una definición de una distribución que aparece en el teorema.

Definición 4.2.1. Se define la distribución ψ_d^k sobre R como:

1. Tomar $\mathbf{y} \leftarrow R^k$.
2. Devolver $(\mathbf{y} - \texttt{Decompress}_q(\texttt{Compress}_q(\mathbf{y}, d), d)) \bmod^{\pm} q$.

Teorema 4.2.1. Sea k un parámetro entero positivo. Sean $\mathbf{s}$, $\mathbf{e}$, $\mathbf{r}$, $\mathbf{e}_1$ y e_2 variables aleatorias con la misma distribución base (como en los algoritmos `Kyber.CPA.KeyGen` y `Kyber.CPA.Enc`). Además, sean $\mathbf{c}_t \leftarrow \psi_{d_t}^k$, $\mathbf{c}_u \leftarrow \psi_{d_u}^k$ y $c_v \leftarrow \psi_{d_v}$. Denotemos

$$\delta = P\left(\|\mathbf{e}^T\mathbf{r} + e_2 + c_v - \mathbf{s}^T\mathbf{e}_1 + \mathbf{c}_t^T\mathbf{r} - \mathbf{s}^T\mathbf{c}_u\|_\infty \geq \lceil q/4 \rfloor\right).$$

Entonces Kyber.CPA es $(1-\delta)$-correcto.

Demostración. El valor de $\mathbf{t}$ en la línea 2 del algoritmo `Kyber.CPA.Enc` podemos escribirlo como

$$\mathbf{t} = \texttt{Decompress}_q(\texttt{Compress}_q(\mathbf{As} + \mathbf{e}, d_t), d_t) = \mathbf{As} + \mathbf{e} + \mathbf{c}_t$$

para algún $\mathbf{c}_t \in R^k$.

A su vez, el valor de $\mathbf{u}$ en el algoritmo `Kyber.CPA.Dec` podemos escribirlo como

$$\mathbf{u} = \texttt{Decompress}_q(\texttt{Compress}_q(\mathbf{A}^T\mathbf{r} + \mathbf{e}_1, d_u), d_u) = \mathbf{A}^T\mathbf{r} + \mathbf{e}_1 + \mathbf{c}_u$$

para algún $\mathbf{c}_u \in R^k$.

Por último, el valor de v también en el algoritmo `Kyber.CPA.Dec` es, para algún $c_v \in R$:

$$\begin{aligned} v &= \texttt{Decompress}_q(\texttt{Compress}_q(\mathbf{t}^T r + e_2 + \lceil q/2 \rfloor \cdot m, d_v), d_v) \\ &= \mathbf{t}^T\mathbf{r} + e_2 + \lceil q/2 \rfloor \cdot m + c_v \\ &= (\mathbf{As} + \mathbf{e} + \mathbf{c}_t)^T\mathbf{r} + e_2 + \lceil q/2 \rfloor \cdot m + c_v \\ &= (\mathbf{As} + \mathbf{e})^T\mathbf{r} + e_2 + \lceil q/2 \rfloor \cdot m + c_v + \mathbf{c}_t^T\mathbf{r} \end{aligned}$$

En los tres casos anteriores podemos suponer que $\mathbf{c}_t$, $\mathbf{c}_u$ y c_v siguen la distribución ψ definida en la definición 4.2.1. Esto se debe a que son de la forma $(\mathbf{y} - \texttt{Decompress}_q(\texttt{Compress}_q(\mathbf{y}, d), d)) \bmod^{\pm} q$, para un $\mathbf{y}$ que podemos considerar uniformemente aleatorio debido a la dificultad del problema MLWE de decisión. En efecto, para el caso de $\mathbf{c}_t$, no podemos diferenciar $\mathbf{As} + \mathbf{e}$ de un vector uniformemente aleatorio. Para el caso de $\mathbf{c}_u$, es $\mathbf{A}^T\mathbf{r} + \mathbf{e}_1$ el que no podemos diferenciar de un vector uniformemente aleatorio. En el caso de c_v, el argumento es algo más complejo. Podemos suponer $\mathbf{t}^T\mathbf{r} + e_2 + \lceil q/2 \rfloor \cdot m$ es pseudo aleatorio haciendo dos consideraciones. En primer lugar, como el mensaje a cifrar puede ser cualquiera, m es aleatorio. Además, $\mathbf{t}^t\mathbf{r} + e_2$ es indistinguible de un número aleatorio por la suposición MLWE y el hecho de que el $\mathbf{t}$ usado resulta de comprimir y descomprimir un vector elegido según una distribución uniforme.

En cualquier caso, usando lo anterior tenemos que

$$v - \mathbf{s}^T\mathbf{u} = \mathbf{e}^T\mathbf{r} + e_2 + \lceil q/2 \rfloor \cdot m + c_v + \mathbf{c}_t^T\mathbf{r} - \mathbf{s}^T\mathbf{e}_1 - \mathbf{s}^T\mathbf{c}_u.$$

Si $\|\mathbf{e}^T\mathbf{r} + e_2 + c_v + \mathbf{c}_t^T\mathbf{r} - \mathbf{s}^T\mathbf{e}_1 - \mathbf{s}^T\mathbf{c}_u\| < \lceil q/4 \rfloor$, lo cual según el enunciado pasa con probabilidad $1 - \delta$, entonces podemos escribir $v - \mathbf{s}^T\mathbf{u} = w + \lceil q/2 \rfloor \cdot m$, con $\|w\|_\infty < \lceil q/4 \rfloor$.

Definiendo $m' = \texttt{Compress}_q(v - \mathbf{s}^T\mathbf{u}, 1)$, tenemos que

$$\lceil q/4 \rfloor \overset{*}{\geq} \|v - \mathbf{s}^T\mathbf{u} - \lceil q/2 \rfloor \cdot m'\|_\infty = \|w + \lceil q/2 \rfloor \cdot m - \lceil q/2 \rfloor \cdot m'\|_\infty.$$

Ahora, por la desigualdad triangular y por el hecho de que $\|w\|_\infty < \lceil q/4 \rfloor$, tenemos

$$\|\lceil q/2 \rfloor \cdot (m - m')\|_\infty < 2 \cdot \lceil q/4 \rfloor.$$

Esto, para todo q impar, como el elegido para Kyber, significa que $m = m'$.

Nota: La desigualdad (*) es algo engorrosa de probar. Para $q = 7681$, el parámetro q elegido en Kyber, una demostración es:

Por construcción, $v - \mathbf{s}^T\mathbf{u} \in R_q$. Así, aplicarle $\texttt{Compress}_q$ significa aplicárselo a cada coeficiente. Probemos que para un polinomio cualquiera $p(X) = b_0 + b_1X + \cdots + b_{n-1}X^{n-1} \in R_q$ se verifica $\lceil q/4 \rfloor \geq \|p - \lceil q/2 \rfloor \cdot \texttt{Compress}_q(p, 1)\|_\infty$. En efecto, $\texttt{Compress}_q(p, 1)$ es un polinomio en R_q, $\texttt{Compress}_q(p, 1) = c_0 + c_1X + \cdots + c_{n-1}X^{n-1}$, donde cada coeficiente está definido por

$$c_k = \lceil (2/q) \cdot b_k \rfloor \bmod^{+} 2.$$

Para el polinomio $p - \lceil q/2 \rfloor \cdot \texttt{Compress}_q(p, 1)$, cuyos coeficientes son $d_k = b_k - \lceil q/2 \rfloor \cdot c_k$, el objetivo es probar que para todo $k = 0, \ldots, n-1$, se tiene $\|d_k\|_\infty = |d_k \bmod^{\pm} q| \leq \lceil q/4 \rfloor$.

Teniendo en cuenta que $\lceil q/2 \rfloor = 3841$ y $\lceil q/4 \rfloor = 1920$ tenemos los siguientes casos.

1. $b_k \in \{0, \ldots, 1920\} \implies c_k = \lceil (2/q) \cdot b_k \rfloor \bmod^+ 2 = 0$ y por tanto $d_k = b_k$, de donde $\|d_k\|_\infty = \|b_k\|_\infty \leq \lceil q/4 \rfloor$.

2. $b_k \in \{1921, \ldots, 3841\} \implies c_k = \lceil (2/q) \cdot b_k \rfloor \bmod^+ 2 = 1$ y por tanto $d_k = b_k - 3841 \in \{-1920, \ldots, 0\}$, de donde $\|d_k\|_\infty \leq 1920 = \lceil q/4 \rfloor$.

3. $b_k \in \{3842, \ldots, 5760 = 3\lceil q/4 \rfloor\} \implies c_k = \lceil (2/q) \cdot b_k \rfloor \bmod^+ 2 = 1$ y por tanto $d_k = b_k - 3841 \in \{1, \ldots, 1919\}$, de donde $\|d_k\|_\infty \leq 1920 = \lceil q/4 \rfloor$.

4. $b_k \in \{5761, \ldots, 7680\} \implies c_k = \lceil (2/q) \cdot b_k \rfloor \bmod^+ 2 = 0$ y por tanto $d_k = b_k$. En este caso hace falta reducir $\bmod^\pm$, obteniendo $d_k \bmod^\pm q \in \{-1920, \ldots, -1\}$, de donde $\|d_k\|_\infty \leq 1920 = \lceil q/4 \rfloor$.

□

Ahora que hemos visto la corrección de `Kyber.CPA`, debemos hacer algunas observaciones sobre su seguridad. En [13] se define un criptosistema modificado `Kyber.CPA'`, simplemente no comprimiendo $\mathbf{As} + \mathbf{e}$ en la línea 4 del algoritmo `Kyber.CPA.KeyGen`, de modo que las claves generadas son $(pk = (\mathbf{As} + \mathbf{e}, \rho), sk = s)$. Esta modificación nos fuerza a quitar también la línea 2 de `Kyber.CPA.Enc`. Este criptosistema modificado se prueba que es CPA seguro bajo la suposición MLWE. En concreto se prueba el siguiente teorema:

Teorema 4.2.2 (Seguridad de `Kyber.CPA'`). Para cualquier adversario $\mathcal{A}$ existe un adversario $\mathcal{B}$ tal que $\mathrm{Adv}^{cpa}_{\texttt{Kyber.CPA'}}(\mathcal{A}) \leq \mathrm{Adv}^{mlwe}_{k+1,k,\eta}(\mathcal{B})$, para k, η parámetros del criptosistema `Kyber.CPA'`.

En la prueba de este teorema se utiliza que la primera parte de la clave pública $\mathbf{t} = \mathbf{As}+\mathbf{e}$ es indistinguible de un vector elegido uniformemente por la suposición MLWE. Esto significa que en la práctica podemos suponer que $\mathbf{t}$ tiene distribución uniforme, y que por tanto a su vez $\mathbf{t}^T\mathbf{r} + e_2$ es indistinguible de un escalar uniforme, también por la suposición MLWE.

En el esquema `Kyber.CPA`, $\mathbf{t} = \texttt{Decompress}_q(\texttt{Compress}_q(\mathbf{As}+\mathbf{e}, d_u), d_u)$, y por tanto ya no tenemos que su distribución sea uniforme en R_q^k, aunque la de $\mathbf{As} + \mathbf{e}$ sí lo sea por la suposición MLWE. Una manera de arreglar esto es añadirle un ruido $\mathbf{e}' \in R_q^k$ tal que $\mathbf{t} + \mathbf{e}'$ sí sea uniforme en R_q^k. Sin embargo, esto añade error al descifrar (para los parámetros de [13], δ pasa de 2^{-142} a 2^{-121}). Por otro lado, no deja de ser extraño crear un término de

error adicional, además de ser complicado crearlo exactamente tal que $\mathbf{t}+\mathbf{e}'$ sea uniforme en R_q^k. Por estos motivos, en [13] se define `Kyber.CPA` como hemos visto, sin añadir el término de error adicional. Esta elección se basa en la creencia de que este aspecto técnico no afecta realmente a la seguridad del sistema, ya que en realidad aplicar las funciones `Compress` y `Decompress` solo altera unos pocos bits.

4.3. El KEM CCA seguro

En esta sección aplicaremos una variante de la tranformación de Fujisaki-Okamoto, prácticamente como aparece en [14], para obtener un KEM IND-CCA seguro o, simplemente, CCA seguro.

Sean $G : \{0,1\}^* \to \{0,1\}^{2\times 256}$ y $H : \{0,1\}^* \to \{0,1\}^{256}$ dos funciones resumen. Entonces nuestro KEM $\texttt{Kyber} = (\texttt{KeyGen}, \texttt{Encaps}, \texttt{Decaps})$ se define como:

- El algoritmo de generación de claves $\texttt{KeyGen}(\cdot)$ es como `Kyber.CPA.KeyGen`, pero la clave secreta sk contiene también la clave pública pk y una semilla aleatoria de 256 bits z:

  1. $\rho,\ \sigma, z \leftarrow \{0,1\}^{256}$.
  2. $\mathbf{A} \sim R_q^{k\times k} := \texttt{Sam}(\rho)$.
  3. $(\mathbf{s}, \mathbf{e}) \sim \beta_\eta^k \times \beta_\eta^k := \texttt{Sam}(\sigma)$.
  4. $\mathbf{t} := \texttt{Compress}_q(\mathbf{As} + \mathbf{e}, d_t)$.
  5. Devolver $(pk := (\mathbf{t}, \rho), sk := (\mathbf{s}, z, \mathbf{t}, \rho))$.

- El algoritmo de encapsulamiento $\texttt{Encaps}(pk = (\mathbf{t}, \rho))$ hace:

  1. $m \leftarrow \{0,1\}^{256}$.
  2. $(\hat{K}, r) := G(H(pk, m))$.
  3. $(\mathbf{u}, v) := \texttt{Kyber.CPA.Enc}((\mathbf{t}, \rho), m; r)$.
  4. $c := (\mathbf{u}, v)$.
  5. $K := H(\hat{K}, H(c))$.
  6. Devolver (c, K).

  donde la notación $\texttt{Kyber.CPA.Enc}(\cdot, \cdot; r)$ significa que r es la semilla que da la aleatoridad del algoritmo `Kyber.CPA.Enc`.

- El algoritmo de desencapsulamiento $\texttt{Decaps}(sk = (\mathbf{s}, z, \mathbf{t}, \rho), c = (\mathbf{u}, v))$ hace:

1. $m' :=$ `Kyber.CPA.Dec`$(\mathbf{s}, (\mathbf{u}, v))$
2. $(\hat{K}', r') := G(H(pk), m')$.
3. $(\mathbf{u}', v') :=$ `Kyber.CPA.Enc`$((\mathbf{t}, \rho), m'; r')$.
4. Si $(\mathbf{u}', v') = (\mathbf{u}, v)$, entonces Devolver $K := H(\hat{K}', H(c))$.
5. Si $(\mathbf{u}', v') \neq (\mathbf{u}, v)$, entonces Devolver $K := H(z, H(c))$.

Estos algoritmos surgen de aplicarle a `Kyber.CPA` la tranformación $FO^{\not\perp}$ de [14], que es la composición de una transformación T y una transformación $U^{\not\perp}$, con una ligera variación. En general, T pasa, con la ayuda de una función resumen G, de un criptosistema CPA seguro a un criptosistema determinístico con una noción de seguridad algo más débil. Por su parte $U^{\not\perp}$ pasa de este criptosistema determinístico al KEM CCA seguro, con la ayuda de la función resumen H, mediante un proceso de recifrado que garantiza la integridad del texto cifrado. El símbolo $\not\perp$ significa que si el proceso de cifrado falla y `Kyber.Decaps` $(\mathbf{u}', v') \neq (\mathbf{u}, v)$, el algoritmo no devuelve el símbolo de rechazo $\perp$, sino que devuelve el valor $K := H(z, c)$, donde z es una semilla aleatoria secreta. Esto recibe el nombre de *rechazo implícito*. La variación que introduce la transformación utilizada en [13] y expuesta en este trabajo es el hecho de resumir pk a $\hat{K}$ en la línea 2 de `Kyber.Encaps`. Esto se hace para dificultar un ataque en que el adversario obtenga información sobre la clave secreta a través del conocimiento de los valores de m que dan errores.

Respecto a la corrección de este KEM, en [13] se prueba que si `Kyber.CPA` es $(1-\delta)$-correcto y G está modelada como un oráculo aleatorio, entonces el KEM `Kyber` también es $(1-\delta)$-correcto. Por su parte, la CCA seguridad se establece con el siguiente teorema, suponiendo que G y H estén modeladas como oráculos aleatorios:

Teorema 4.3.1. Para cualquier adversario clásico $\mathcal{A}$ que hace como mucho q_{RO} preguntas a los oráculos aleatorios de G y H, y q_D preguntas al oráculo de descifrado, entonces existe un adversario $\mathcal{B}$ tal que

$$\mathrm{Adv}^{cca}_{\mathrm{Kyber}}(\mathcal{A}) \leq 3\mathrm{Adv}^{cpa}_{\mathrm{Kyber.CPA}}(\mathcal{B}) + q_{RO}\delta + \frac{3q_{RO}}{2^{256}}$$

Para el caso cuántico; esto es, en el modelo del oráculo aleatorio cuántico, el análisis es algo más complicado, pero también se llega a probar que, al menos asintóticamente, `Kyber` es CCA seguro.

4.4. El criptosistema CCA seguro

Como hicimos para desarrollar los criptosistemas RSA y ElGamal en el capítulo 1, nos serviremos de un cifrado simétrico 1CCA seguro, $SKE = (E, D)$, para construir nuestro criptosistema CCA seguro híbrido. El cifrado simétrico debe tener como espacio de claves $\mathcal{K} = \{0,1\}^{256}$ y como espacio de mensajes $\{0,1\}^*$. Permitiremos que el algoritmo D devuelva un mensaje o un símbolo de rechazo $\perp$. El criptosistema de clave pública resultante recibe el nombre de $\texttt{Kyber.Hybrid} = (\texttt{KeyGen}, \texttt{Enc}, \texttt{Dec})$, y viene definido como sigue:

- El algoritmo de generación de claves $\texttt{KeyGen}(\cdot)$ hace:

  1. $(pk := (\mathbf{t}, \rho), sk := (\mathbf{s}, z, \mathbf{t}, \rho)) \leftarrow \texttt{Kyber.KeyGen}(\cdot)$
  2. Devolver (pk, sk).

- El algoritmo de cifrado $\texttt{Enc}(pk = (\mathbf{t}, \rho), m)$ hace

  1. $(c, K) \leftarrow \texttt{Kyber.Encaps}(pk)$
  2. $c' := E(K, m)$
  3. Devolver $c'' := (c, c')$.

- El algoritmo de descifrado $\texttt{Dec}(sk = (\mathbf{s}, z, \mathbf{t}, \rho), c'' = (c, c'))$ hace:

  1. $K := \texttt{Kyber.Decaps}(sk, c)$
  2. Devolver $m := D(K, c')$.

Este criptosistema es solo una instancia del procedimiento general para obtener criptosistemas de clave pública CCA seguros a partir de KEM CCA seguros y cifrados simétricos 1CCA seguros. En [15] se prueba que este tipo de construcciones son correctas y CCA seguras.

4.5. Parámetros de Kyber

En la tabla 4.1 vemos los parámetros que se le da a Kyber en [13]. Se definen tres sets de parámetros, Kyber (el recomendado), Light (algo más inseguro) y Paranoid (más seguro). Para cada set de parámetros, vemos también los tamaños de las claves pública y privada para el KEM `Kyber`.

Nivel de seguridad	k	η	(d_u, d_v, d_t)	δ	sk (bytes)	pk (bytes)
Kyber	3	4	$(11, 3, 11)$	2^{-142}	2400	1088
Light	2	5	$(11, 3, 11)$	2^{-145}	2048	736
Paranoid	4	3	$(11, 3, 11)$	2^{-169}	2752	1440

Cuadro 4.1: Parámetros usados en [13] y tamaños de claves y texto cifrado

Como ya se ha mencionado, operamos siempre sobre los mismos anillos, y los diferentes niveles de seguridad y corrección se consiguen variando únicamente algunos parámetros. En particular, aumentar k aumenta la dimensión del retículo que utilizamos. Vemos también que los tamaños de claves y texto cifrado son bastante reducidos, lo que implica una gran eficiencia del criptosistema.

Resumen del capítulo

En este capítulo se ha explicado Kyber, la propuesta de CRYSTALS para el concurso del NIST para KEM, cumpliéndose así uno de los objetivos del trabajo. Hemos visto algunos preliminares, para después meternos de lleno en la descripción de un criptosistema de clave pública CPA seguro. Este criptosistema CPA seguro ha resultado no ser muy diferente al criptosistema basado en el problema LWE que vimos en el capítulo 3, siendo la principal ventaja del nuevo el estar basado en retículos estructurados, lo que le da una eficiencia mucho mayor. A partir de este criptosistema hemos conseguido un KEM CCA seguro, que es la propuesta para el concurso del NIST propiamente dicha. Por último, este KEM nos ha servido para describir un criptosistema CCA seguro.

Capítulo 5

Dilithium. Un esquema de firma digital SUF-CMA seguro

En este capítulo se desarrollará Dilithium, la propuesta de CRYSTALS para el concurso del NIST para esquemas de firma digital, siguiendo la línea de [16]. Como ya se ha mencionado, Dilithium es un esquema de firma digital cuya seguridad se basa en la dificultad de resolver problemas de retículos. Como veremos, estos problemas son los problemas MLWE, MSIS y una variante de este último. Además de su seguridad, algunas de las características que diferencian a Dilithium son:

- **Es simple de implementar**: Como vimos en el capítulo 3, una forma usual de conseguir esquemas de firma digital seguros basándonos en problemas de retículos es aleatorizar la firma mediante una distribución normal. Generar estas firmas de modo que efectivamente sean seguras no es en absoluto trivial, además de ser costoso computacionalmente. Dilithium esquiva este problema utilizando solamente distribuciones uniformes, dando lugar al mismo tiempo a un esquema seguro y menos costoso de implementar.

- **Tamaño de clave pública y firma mínimos:** Dilithium tiene la menor combinación de tamaños de la clave pública y de la firma de entre todos los esquemas propuestos, para el concurso del NIST, con el mismo nivel de seguridad.

- **Flexibilidad para variar el nivel de seguridad:** Al igual que en Kyber, trabajaremos siempre sobre un anillo $\mathbb{Z}_q/(X^n+1)$, en este caso con $q = 2^{23} - 2^{13} + 1$ y $n = 256$.

Optimizaremos las operaciones en este anillo, y cambiar el nivel de seguridad exigido simplemente significará hacer más operaciones. No hará faltar cambiar de anillo y encontrar algoritmos de cálculo eficientes en el nuevo anillo.

5.1. Preliminares

De nuevo, la construcción de Dilithium requiere de muchos detalles técnicos, que veremos a lo largo de esta sección.

5.1.1. Anillos

Trabajaremos con los anillos $R = \mathbb{Z}[X]/(X^n + 1)$ y $R_q = \mathbb{Z}_q[X]/(X^n + 1)$. Como ya se ha mencionado, usaremos $n = 256$ y el primo $q = 8380417 = 2^{23} - 2^{13} + 1$. Como en el caso de Kyber, denotaremos los elementos de los anillos R, R_q, $\mathbb{Z}$ y $\mathbb{Z}_q$ con letras normales, los vectores de elementos de anillos en letra minúscula negrita, y las matrices en letra mayúscula negrita. Para un vector $\mathbf{v}$ y una matriz $\mathbf{A}$, denotaremos $\mathbf{v}^T$ ó $\mathbf{A}^T$ a su traspuesto.

Para las normas del capítulo anterior, definimos los siguientes conjuntos:

$$S_\eta = \{w \in R : \|w\|_\infty \leq \eta\}$$
$$\tilde{S}_\eta = \{w \bmod^{\pm} 2\eta : w \in R\}$$

Estos dos conjuntos son muy similares salvo por el hecho de que $\tilde{S}_\eta$ no tiene elementos con coeficientes $-\eta$.

5.1.2. Funciones resumen

Dilithium utiliza varias funciones resumen, que llevan elementos de $\{0,1\}^*$ (cadenas de bits de longitud arbitraria) a conjuntos de varias formas. A continuación se ofrece una descripción somera de estas funciones:

Función `SampleInBall`(ρ)

Sea B_τ el conjunto de elementos de R que tienen τ coeficientes que son -1 ó 1, y el resto son 0. Se tiene $|B_\tau| = 2^\tau \binom{256}{\tau}$. La función `SampleInBall`$(\rho)$ tomará $\rho \in \{0,1\}^*$ y nos devolverá un polinomio aleatorio de B_τ. Se usa ρ como semilla para la aleatoridad de la función.

Funciones $\texttt{ExpandA}(\rho)$ **y** $\texttt{ExpandS}(\rho')$

La función `ExpandA` lleva una semilla $\rho \in \{0,1\}^{256}$ a una matriz aleatoria $A \in R_q^{k \times l}$. Por su parte, la función `ExpandS` lleva una semilla $\rho' \in \{0,1\}^{512}$ a dos vectores $(\mathbf{s}_1, \mathbf{s}_2) \in S_\eta^l \times S_\eta^k$.

Función $\texttt{ExpandMask}(\rho, \kappa)$

Esta función genera determinísticamente la aleatoridad del proceso de firma. Toma $\rho' \in \{0,1\}^{512}$ y un número κ y devuelve un vector $\mathbf{y} \in \tilde{S}_{\gamma_1}^l$, con γ_1 un parámetro.

Función resumen H

Será la función que usemos para resumir los mensajes a firmar. Se trata de una función de salida extendible (XOF) basada en la función estándar SHAKE-256.

5.1.3. Funciones que obtienen bits de mayor y menor orden

Para reducir el tamaño de la clave pública, Dilithium utiliza algoritmos que extraen los bits de mayor y menor orden de elementos en $\mathbb{Z}_q$. El objetivo es que, dado un $r \in \mathbb{Z}_q$ arbitrario y un $z \in \mathbb{Z}_q$ pequeño, queremos obtener los bits de mayor orden de $r+z$ sin tener que almacenar z.

La primera forma en que romperemos un elemento en $\mathbb{Z}_q$ en sus bits de mayor y menor orden es el algoritmo $\texttt{Power2Round}_q(r, d)$. Este algoritmo nos devuelve r_1 y r_0 tales que $r = r_1 2^d + r_0$. Estos números se obtienen haciendo $r_0 = r \bmod^{\pm} 2^d$ y $r_1 = \frac{r - r_0}{2^d}$, donde $\bmod^{\pm}$ es la reducción modular definida en el capítulo anterior.

Otra forma de hacerlo será la función $\texttt{Decompose}_q(r, \alpha)$. Seleccionamos α un divisor par de $q-1$ y escribimos $r = r_1\alpha + r_0$ del mismo modo que antes. En principio, los posibles $r_1\alpha$ están en $\{0, \alpha, 2\alpha, \ldots, q-1\}$. Como la distancia entre $q-1$ y 0 es 1, quitamos $q-1$ de este conjunto y simplemente redondeamos los r involucrados a 0. Esto lo que hace en el peor de los casos es aumentar el resto r_0 en una unidad. La función $\texttt{Decompose}_q(r, \alpha)$ devuelve r_1 y r_0 así obtenidos. r_0 representará los bits de menor orden y r_1 los de mayor. Por simplicidad en la notación definimos las funciones $\texttt{HighBits}_q(r, \alpha)$ y $\texttt{LowBits}_q(r, \alpha)$ como las funciones que devuelven r_1 y r_0, respectivamente.

Para nuestro objetivo de extraer los bits de mayor orden de $r + z$ sin almacenar z se definen las funciones $\texttt{MakeHint}_q(z, r, \alpha)$ y $\texttt{UseHint}_q(h, r, \alpha)$, que respectivamente crean y

utilizan una pista h para obtener los bits de mayor orden de $r+z$. $\texttt{MakeHint}_q(z, r, \alpha)$ calcula $r_1 = \texttt{HighBits}_q(r, \alpha)$ y $v_1 = \texttt{HighBits}_q(r + z, \alpha)$, y devuelve un 1 si $r_1 \neq v_1$ y 0 en otro caso. Por su parte, $\texttt{UseHint}_q(h, r, \alpha)$ calcula $(r_1, r_0) := \texttt{Decompose}_q(r, \alpha)$. Si la pista h es 0, el algoritmo devuelve r_1. Si la pista h es 1 se diferencian dos casos. Si $r_0 > 0$ se devuelve $(r_1 + 1) \bmod^{\pm} m$, para $m := (q-1)/\alpha$, mientras que si $r_0 \leq 0$ se devuelve $(r_1 - 1) \bmod^{\pm} m$.

Los siguientes lemas gobiernan el funcionamiento de estas funciones, y serán de gran utilidad para probar la corrección y la seguridad del esquema de firma digital. Cuando una de estas funciones se aplique a un vector, entenderemos que se aplica la función componente a componente. Cuando se aplique a un polinomio, se aplicará coeficiente a coeficiente.

Lema 1. Sean q y α enteros positivos con $q > 2\alpha$, $q \equiv 1 \bmod \alpha$ y α par. Sean $\mathbf{r}$ y $\mathbf{z}$ vectores de elementos de R_q, con $\|\mathbf{z}\|_\infty \leq \alpha/2$, y sean $\mathbf{h}$, $\mathbf{h}'$ vectores de bits. Entonces las funciones $\texttt{HighBits}_q$, $\texttt{MakeHint}_q$ y $\texttt{UseHint}_q$ satisfacen las siguientes propiedades:

1. $\texttt{UseHint}_q(\texttt{MakeHint}_q(\mathbf{z}, \mathbf{r}, \alpha), \mathbf{r}, \alpha) = \texttt{HighBits}_q(\mathbf{r} + \mathbf{z}, \alpha)$

2. Sea $\mathbf{v}_1 = \texttt{UseHint}_q(\mathbf{h}, \mathbf{r}, \alpha)$. Entonces $\|\mathbf{r} - \mathbf{v}_1 \cdot \alpha\|_\infty \leq \alpha + 1$. Más aún, si el número de unos en $\mathbf{h}$ es w, entonces todos los coeficientes de $\mathbf{r} - \mathbf{v}_1 \cdot \alpha$ salvo a lo sumo w de ellos tendrán magnitud como mucho $\alpha/2$ tras aplicarles una reducción centrada módulo q.

3. Para cualesquiera $\mathbf{h}$, $\mathbf{h}'$, si $\texttt{UseHint}_q(\mathbf{h}, \mathbf{r}, \alpha) = \texttt{UseHint}_q(\mathbf{h}', \mathbf{r}, \alpha)$, entonces $\mathbf{h} = \mathbf{h}'$.

Lema 2. Si $\|\mathbf{s}\|_\infty \leq \beta$ y $\|\texttt{LowBits}_q(\mathbf{r}, \alpha)\|_\infty < \alpha/2 - \beta$, entonces

$$\texttt{HighBits}_q(\mathbf{r}, \alpha) = \texttt{HighBits}_q(\mathbf{r} + \mathbf{s}, \alpha)$$

Lema 3. Sean $(\mathbf{r}_1, \mathbf{r}_0) = \texttt{Decompose}_q(\mathbf{r}, \alpha)$ y $(\mathbf{w}_1, \mathbf{w}_0) = \texttt{Decompose}_q(\mathbf{r}+\mathbf{s}, \alpha)$, con $\|\mathbf{s}\|_\infty \leq \beta$. Entonces

$$\|\mathbf{s} + \mathbf{r}_0\|_\infty < \alpha/2 - \beta \iff \mathbf{w}_1 = \mathbf{r}_1 \wedge \|\mathbf{w}_0\|_\infty < \alpha/2 - \beta$$

5.2. El esquema de firma digital

5.2.1. Un esquema parecido menos eficiente

Para entender correctamente cómo funciona Dilithium, es conveniente introducir antes otro esquema de firma digital basado en la misma idea, menos eficiente pero más sencillo de entender. Denotaré a este esquema de firma digital `Idea`=(`Gen`, `Sign`, `Verify`) y en

adelante me referiré a estos algoritmos como `Idea.Algoritmo`. Estos algoritmos se definen como sigue:

- El algoritmo de generación de claves $\texttt{Gen}(\cdot)$ hace:

  1. $\mathbf{A} \leftarrow R_q^{k \times l}$.
  2. $(\mathbf{s}_1, \mathbf{s}_2) \leftarrow S_\eta^l \times S_\eta^k$.
  3. $\mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$.
  4. Devolver $(pk = (\mathbf{A}, \mathbf{t}), sk = (\mathbf{A}, \mathbf{t}, \mathbf{s}_1, \mathbf{s}_2))$.

- El algoritmo de firma $\texttt{Sign}(sk, M)$ hace:

  1. $\mathbf{z} :=\perp$.
  2. Mientras que $\mathbf{z} =\perp$, hacer:
     a) $\mathbf{y} \leftarrow S_{\gamma_1 - 1}^l$.
     b) $\mathbf{w}_1 := \texttt{HighBits}_q(\mathbf{A}\mathbf{y}, 2\gamma_2)$.
     c) $c := \mathrm{H}(M||\mathbf{w}_1) \in B_\tau$.
     d) $\mathbf{z} := \mathbf{y} + c\mathbf{s}_1$.
     e) Si $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ ó $\|\texttt{LowBits}_q(\mathbf{A}\mathbf{y} - c\mathbf{s}_2, 2\gamma_2)\|_\infty \geq \gamma_2 - \beta$, entonces $\mathbf{z} :=\perp$.
  3. Devolver $\sigma = (\mathbf{z}, c)$.

- El algoritmo de verificación $\texttt{Verify}(pk, M, \sigma = (\mathbf{z}, c))$ hace:

  1. $\mathbf{w}_1' := \texttt{HighBits}_q(\mathbf{A}\mathbf{z} - c\mathbf{t}, 2\gamma_2)$.
  2. Devolver 1 si $\|\mathbf{z}\|_\infty < \gamma_1 - \beta \wedge c = \mathrm{H}(M||\mathbf{w}_1')$, y 0 en otro caso.

¿Qué es lo que se esta haciendo? Examinemos estos algoritmos uno a uno. En primer lugar, el algoritmo `Idea.Gen` toma $\mathbf{A}$ una matriz $k \times l$ con un polinomio de R_q en cada entrada, y dos vectores $\mathbf{s}_1$ y $\mathbf{s}_2$ tales que cada elemento suyo es un polinomio en R_q con coeficientes a lo sumo η. Después se calcula el vector $\mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$ y se toma como clave pública junto a $\mathbf{A}$. Por los problemas de retículos conocidos, esperaremos que dados $\mathbf{t}$ y $\mathbf{A}$ sea difícil recuperar $\mathbf{s}_1$ y $\mathbf{s}_2$.

El algoritmo `Idea.Sign`, por su parte, lo primero que hace es generar un vector 'enmascarante' $\mathbf{y}$. Posteriormente se calcula $\mathbf{A}\mathbf{y}$ y se extraen sus bits de mayor orden. En

concreto, para cada coeficiente w de los polinomios del vector escribimos, como hemos visto en la sección 5.1, $w = w_1 \cdot 2\gamma_2 + w_0$, con $|w_0| \leq \gamma_2$. $\mathbf{w}_1$ será el vector de todos los w_1's. Después se crea c como un resumen del mensaje M y de $\mathbf{w}_1$. Este c es un polinomio de R_q con exactamente τ unos o menos unos y el resto ceros. Una posible firma será entonces $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$. Si esta firma satisface las dos condiciones de la línea 2e del algoritmo, entonces no será una firma válida y el proceso iterativo comienza otra vez. La primera condición se pone por motivos de la seguridad del esquema, y la segunda por motivos de seguridad y también de corrección.

Por último, el algoritmo `Idea.Verify` calcula los bits de mayor orden de $\mathbf{Az} - c\mathbf{t}$ y los guarda en $\mathbf{w}_1'$. El algoritmo acepta la firma si todos sus coeficientes son menores que $\gamma_1 - \beta$ (porque como hemos visto en el algoritmo `Idea.Sign` una firma válida tiene que satisfacer esta condición) y si c es el resumen del mensaje y $\mathbf{w}_1'$. ¿Por qué funciona esto último? Es decir, ¿por qué $\mathbf{w}_1 = \texttt{HighBits}(\mathbf{Ay}, 2\gamma_2) = \texttt{HighBits}(\mathbf{Az} - c\mathbf{t}, 2\gamma_2) = \mathbf{w}_1'$? En primer lugar, por construcción $\mathbf{Az} - c\mathbf{t} = \mathbf{Ay} - c\mathbf{s}_2$. Además, $\texttt{HighBits}(\mathbf{Ay}, 2\gamma_2) = \texttt{HighBits}(\mathbf{Ay} - c\mathbf{s}_2, 2\gamma_2)$ pues por la línea 2e del algoritmo `Idea.Sign` una firma válida cumplirá $\|\texttt{LowBits}(\mathbf{Ay} - c\mathbf{s}_2, 2\gamma_2)\|_\infty < \gamma_2 - \beta$. El parámetro β se elige de modo que sea el máximo valor de los coeficientes de $c\mathbf{s}_i$, de modo que tenemos que todos los coeficientes de $c\mathbf{s}_2$ son menores que β. Así, sumar $c\mathbf{s}_2$ a $\mathbf{Ay} - c\mathbf{s}_2$ no es suficiente para cambiar sus bits de mayor orden.

5.2.2. El esquema de firma digital Dilithium

El esquema del apartado anterior es bastante ineficiente, siendo la ineficiencia más evidente que la clave pública consiste en una matriz $k \times l$ de polinomios. La solución que adopta Dilithium es generar esta matriz $\mathbf{A}$ a partir de una semilla ρ, de modo que la clave pública será $(\rho, \mathbf{t})$.

Además, se encuentra un método para que $\mathbf{t}$ ocupe aproximadamente la mitad de bits, con el coste de incrementar el tamaño de la firma en no más de 100 bits. Básicamente se observa que en el esquema de firma digital `Idea`, los bits de menor orden de $\mathbf{t}$ no influyen mucho en el proceso de verificación, así que podemos no meterlos en la clave pública. Para compensar los posibles inconvenientes que esto produzca al verificador, la firma deberá contener algunas 'pistas'.

Otra ventaja de Dilithium es que se da la opción de que el proceso sea determinístico.

Esto se hará añadiendo una semilla a la clave secreta y usando esta semilla junto con el mensaje para producir la aleatoridad del vector enmascarante $\mathbf{y}$. El interés de esto se debe a que, cuando el esquema es sometido a ataques cuánticos, el esquema es más seguro cuantas menos firmas de un mismo mensaje vea el atacante. Por este motivo, veremos que de hecho el esquema 'predeterminado' de Dilithium utiliza esta versión determinística.

También se hacen otros cambios, como hacer el resumen del mensaje fuera del bucle, de forma que no tengamos que repetirlo en cada iteración, y meter un parámetro κ que cambia en cada iteración, asegurando que si el algoritmo de firma falla no se repita un proceso idéntico en la siguiente iteración. A continuación se presenta el esquema definitivo:

- El algoritmo de generación de claves $\texttt{Dilithium.Gen}(\cdot)$hace:

  1. $\xi \leftarrow \{0,1\}^{256}$.
  2. $(\rho, \rho', K) \in \{0,1\}^{256} \times \{0,1\}^{512} \times \{0,1\}^{256} := \mathrm{H}(\xi)$.
  3. $\mathbf{A} \in R_q^{k \times l} := \texttt{ExpandA}(\rho)$.
  4. $(\mathbf{s}_1, \mathbf{s}_2) \in S_\eta^l \times S_\eta^k := \texttt{ExpandS}(\rho')$.
  5. $\mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$.
  6. $(\mathbf{t}_1, \mathbf{t}_0) := \texttt{Power2Round}_q(\mathbf{t}, d)$.
  7. $tr \in \{0,1\}^{256} := \mathrm{H}(\rho || \mathbf{t}_1)$.
  8. Devolver $(pk = (\rho, \mathbf{t}_1), sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0))$.

- El algoritmo de firma $\texttt{Dilithium.Sign}(sk, M)$ hace:

  1. $\mathbf{A} \in R_q^{k \times l} := \texttt{ExpandA}(\rho)$.
  2. $\mu \in \{0,1\}^{512} := \mathrm{H}(tr || M)$.
  3. $\kappa := 0, (\mathbf{z}, \mathbf{h}) := \perp$.
  4. $\rho' \in \{0,1\}^{512} := \mathrm{H}(K || \mu)$ (o, para firma aleatorizada, $\rho' \leftarrow \{0,1\}^{512}$).
  5. Mientras $(\mathbf{z}, \mathbf{h}) := \perp$, hacer:
     a) $\mathbf{y} \in \tilde{S}_{\gamma_1}^l := \texttt{ExpandMask}(\rho', \kappa)$.
     b) $\mathbf{w} := \mathbf{A}\mathbf{y}$.
     c) $\mathbf{w}_1 := \texttt{HighBits}_q(\mathbf{w}, 2\gamma_2)$.
     d) $\tilde{c} \in \{0,1\}^{256} := \mathrm{H}(\mu || \mathbf{w}_1)$.
     e) $c \in B_\tau := \texttt{SampleInBall}(\tilde{c})$.
     f) $\mathbf{z} := \mathbf{y} + c\mathbf{s}_1$.
     g) $\mathbf{r}_0 := \texttt{LowBits}_q(\mathbf{w} - c\mathbf{s}_2, 2\gamma_2)$.

h) Si $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ o $\|\mathbf{r}_0\|_\infty \geq \gamma_2 - \beta$, entonces $(\mathbf{z}, \mathbf{h}) := \perp$.

En otro caso:

- $\mathbf{h} := \texttt{MakeHint}_q(-c\mathbf{t}_0, \mathbf{w} - c\mathbf{s}_2 + c\mathbf{t}_0, 2\gamma_2)$.
- Si $\|c\mathbf{t}_0\|_\infty \geq \gamma_2$ ó el número de unos en $\mathbf{h}$ es mayor que w, entonces $(\mathbf{z}, \mathbf{h}) := \perp$.

i) $\kappa = \kappa + l$.

6. Devolver $\sigma = (\tilde{c}, \mathbf{z}, \mathbf{h})$.

- El algoritmo de verificación $\texttt{Dilithium.Verify}(pk = (\rho, \mathbf{t}_1), M, \sigma = (\tilde{c}, \mathbf{z}, \mathbf{h}))$ hace:

1. $\mathbf{A} \in R_q^{k \times l} := \texttt{ExpandA}(\rho)$.
2. $\mu \in \{0,1\}^{512} := \mathrm{H}(\mathrm{H}(\rho||\mathbf{t}_1)||M)$.
3. $c := \texttt{SampleInBall}(\tilde{c})$.
4. $\mathbf{w}_1' := \texttt{UseHint}_q(\mathbf{h}, \mathbf{Az} - c\mathbf{t}_1 \cdot 2^d, 2\gamma_2)$.
5. Devolver 1 si $(\|\mathbf{z}\|_\infty < \gamma_1 - \beta) \wedge (\tilde{c} = \mathrm{H}(\mu||\mathbf{w}_1')) \wedge (\#1\text{'s en } \mathbf{h} \leq w)$, y 0 en otro caso.

Probemos la corrección de este esquema. Supongamos que le pasamos al algoritmo `Dilithium.Verify` un par $(M, \sigma = (\tilde{c}, \mathbf{z}, \mathbf{h}))$ válido. Esta firma válida habrá sido producida en una iteración en que $\|c\mathbf{t}_0\|_\infty < \gamma_2$. Entonces, por el lema 1,

$$\texttt{UseHint}_q(\mathbf{h}, \mathbf{w} - c\mathbf{s}_2 + c\mathbf{t}_0, 2\gamma_2) = \texttt{HighBits}_q(\mathbf{w} - c\mathbf{s}_2, 2\gamma_2)$$

Además, $\mathbf{w} = \mathbf{Ay}$ y $\mathbf{t} = \mathbf{As}_1 + \mathbf{s}_2$, por lo que

$$\mathbf{w} - c\mathbf{s}_2 = \mathbf{Ay} - c\mathbf{s}_2 = \mathbf{A}(\mathbf{z} - c\mathbf{s}_1) - c\mathbf{s}_2 = \mathbf{Az} - c\mathbf{t}$$

Por construcción de $\mathbf{t}_1$ y $\mathbf{t}_0$, tenemos también que $\mathbf{w} - c\mathbf{s}_2 + c\mathbf{t}_0 = \mathbf{Az} - c\mathbf{t}_1 \cdot 2^d$. De este modo, en el algoritmo `Dilithium.Verify` se calcula

$$\texttt{UseHint}_q(\mathbf{h}, \mathbf{Az} - c\mathbf{t}_1 \cdot 2^d, 2\gamma_2) = \texttt{HighBits}_q(\mathbf{w} - c\mathbf{s}_2, 2\gamma_2)$$

Por otro lado, por definición de β se verifica que $\|c\mathbf{s}_2\|_\infty \leq \beta$. Además, en la línea 5h de `Dilithium.Sign` se verifica que $\texttt{LowBits}_q(\mathbf{w} - c\mathbf{s}_2, 2\gamma_2) < \gamma_2 - \beta$. Estas dos condiciones, junto con el lema 2 implican que

$$\texttt{HighBits}_q(\mathbf{w} - c\mathbf{s}_2, 2\gamma_2) = \texttt{HighBits}_q(\mathbf{w} - c\mathbf{s}_2 + c\mathbf{s}_2, 2\gamma_2) = \texttt{HighBits}_q(\mathbf{w}, 2\gamma_2) \stackrel{\text{def}}{=} \mathbf{w}_1$$

Así, el $\mathbf{w}_1'$ calculado en `Dilithium.Verify` coincide con el $\mathbf{w}_1$ calculado en `Dilithium.Sign`, y el proceso de verificación acepta.

5.2.3. Parámetros de Dilithium

¿Qué parámetros son los elegidos en [16] para este esquema de firma digital? Dependerá del nivel de seguridad exigido. En la siguiente tabla, se ven los parámetros utilizados dependiendo del nivel de seguridad del NIST exigido. Intuitivamente, a más nivel de seguridad más seguro es el esquema de firma digital, y más tipos de ataques resiste. Para cada nivel de seguridad se dan también los tamaños de clave pública, clave privada y firma, calculados según las fórmulas de [16].

	Nivel de seguridad		
Parámetro	1	3	5
q	8380417	8380417	8380417
d (bits quitados a $\mathbf{t}$)	13	13	13
τ (# de 1's en c)	39	49	60
γ_1 (rango de los coeficientes de $\mathbf{y}$)	2^{17}	2^{19}	2^{19}
γ_2 (para separar bits de mayor y menor orden)	$\frac{q-1}{88}$	$\frac{q-1}{32}$	$\frac{q-1}{32}$
(k, l) (dimensiones de $\mathbf{A}$)	(4,4)	(6,5)	(8,7)
η (rango de la clave secreta)	2	4	2
β (rango de los coeficientes de $c\mathbf{s}_i$)	78	196	120
w (máx. # de 1's en $\mathbf{h}$)	80	55	75
Tamaños (en bytes)			
Clave pública	1312	1952	2592
Clave secreta	2016	2944	3904
Firma	2420	3293	4595

Cuadro 5.1: Parámetros elegidos en [16]

En la tabla vemos dos de las características fundamentales de Dilithium. Por un lado, para aumentar el nivel de seguridad solo aumentamos algunos parámetros, pero las operaciones se hacen siempre en los mismos anillos. Por otro lado, vemos que las claves pública y privada, así como la firma digital, tienen tamaños bastante manejables.

5.2.4. Seguridad de Dilithium

En [16] se prueba la SUF-CMA seguridad (fuertemente existencialmente infalsificable bajo ataques de mensaje escogido) de Dilithium en base a 3 problemas de retículos sobre módulos, el problema MLWE, el problema MSIS y el problema SelfTargetMSIS, que es un problema cuya dureza se basa en la dureza del MSIS y de la seguridad de la función resumen utilizada H. Estos problemas se formalizan de la siguiente manera:

El problema MLWE

Se define igual que en el capítulo 4, pero en esta ocasión la distribución elegida para generar los vectores $\mathbf{s}$ y $\mathbf{e}$ no será la distribución β_η, sino una distribución cualquiera D sobre R_q. Se define entonces la ventaja de un adversario $\mathcal{A}$, denotada $\mathrm{Adv}^{mlwe}_{m,k,D}(\mathcal{A})$ como en el juego 4.1.2, pero haciendo el cambio mencionado.

El problema MSIS

Dada una matriz aleatoria uniforme $\mathbf{A} \leftarrow R_q^{m\times k}$ y un parámetro γ, el problema $\mathrm{MSIS}_{m,k,\gamma}$ consiste en encontrar un vector $\mathbf{y} \in R_q^m$ tal que $0 < \|\mathbf{y}\|_\infty \leq \gamma$, y $[\ \mathbf{I} \mid \mathbf{A}\ ]\mathbf{y} = 0$, donde por $[\ \mathbf{I} \mid \mathbf{A}\ ]$ entendemos que si $k < m$ se completa $\mathbf{A}$ con una matriz $m \times (m-k)$ con todo ceros salvo unos en la diagonal principal. Se define la ventaja de $\mathcal{A}$, $\mathrm{Adv}^{msis}_{m,k,\eta}(\mathcal{A})$ como la probabilidad de que el adversario $\mathcal{A}$ resuelva este problema.

El problema SelfTargetMSIS

Sea H: $\{0,1\}^* \to B_\tau$ una función resumen. Dada una matriz $\mathbf{A} \leftarrow R_q^{m\times k}$, el problema SelfTargetMSIS consiste en encontrar un vector $\mathbf{y} := \begin{bmatrix} \mathbf{r} \\ c \end{bmatrix}$ y un $\mu \in \{0,1\}^*$ tales que $0 \leq \|\mathbf{y}\|_\infty \leq \gamma$, y $\mathrm{H}(\mu \,||\, [\ \mathbf{I} \mid \mathbf{A}\ ]) = c$.

Se define la ventaja de un adversario $\mathcal{A}$, $\mathrm{Adv}^{selftargetmsis}_{\mathrm{H},m,k,\gamma}(\mathcal{A})$, como la probabilidad de que $\mathcal{A}$ resuelva este problema.

En un contexto en que el algoritmo $\mathcal{A}$ es clásico y la función H es un oráculo aleatorio, existe una reducción del problema MSIS al problema SelfTargetMSIS; es decir, resolver SelfTargetMSIS implica resolver MSIS. En este contexto, la SUF-CMA seguridad de Dilithium se puede probar simplemente basándonos en la dureza de los problemas MLWE y MSIS. Sin embargo, cuando el adversario $\mathcal{A}$ es capaz de hacer ataques cuánticos esta reducción

deja de ser válida. Dilithium busca ser un esquema de firma digital postcuántico, así que este es el contexto que nos interesa. En este caso, tenemos el siguiente teorema:

Teorema 5.2.1 (Seguridad de Dilithium). Supongamos que H sea una función resumen modelada como un oráculo aleatorio cuántico. Dado un adversario $\mathcal{A}$, existen adversarios $\mathcal{A}$, $\mathcal{C}$ y $\mathcal{D}$ tales que

$$\mathrm{Adv}^{SUF-CMA}_{Dilithium}(\mathcal{A}) \leq \mathrm{Adv}^{mlwe}_{k,l,D}(\mathcal{B}) + \mathrm{Adv}^{selftargetmsis}_{\mathrm{H},k,l+1,\xi}(\mathcal{C}) + \mathrm{Adv}^{msis}_{k,l,\xi'}(\mathcal{D}) + 2^{-254}$$

para D una distribución uniforme sobre S_η y

$$\xi = \mathrm{máx}\{\gamma_1 - \beta, 2\gamma_2 + 1 + 2^{d-1} \cdot \tau\}$$
$$\xi' = \mathrm{máx}\{2(\gamma_1 - \beta), 4\gamma_2 + 2\}$$

Resumen del capítulo

En este capítulo hemos alcanzado también uno de los objetivos del TFG: describir el funcionamiento de Dilithium, la propuesta de CRYSTALS para el concurso del NIST para firma digital. Como en el capítulo 4, hemos comenzado viendo algunos preliminares, para después meternos de lleno en la descripción de la propuesta. En este caso, lo que hemos hecho ha sido describir un esquema parecido menos eficiente, explicando por qué no lo es y cómo se mejora en la versión final. Hemos tratado también los problemas sobre retículos en los que se basa la seguridad del esquema, y hemos visto un teorema que garantiza su seguridad incluso en un contexto cuántico.

Conclusión

En este Trabajo Fin de Grado se han desarrollado las propuestas Kyber y Dilithium para el concurso del NIST sobre criptografía postcuántica. Como se ha dicho en la introducción, el TFG se ha desarrollado desde cero, sin conocimientos previos de criptografía ni de computación cuántica. Así, gran parte del trabajo que he hecho ha sido documentarme, leyendo numerosas fuentes y familiarizándome con conceptos y maneras de formalizar propiedades que desconocía, como las nociones de seguridad y su formalización mediante juegos.

En el primer capítulo se ha comenzado viendo las definiciones y resultados básicos sobre criptografía de clave pública, para después profundizar en las técnicas criptográficas basadas en la dificultad de factorizar números enteros y el problema del logaritmo discreto. La importancia de este capítulo radica en que se dan las definiciones sobre las que se asienta todo lo demás. Además, las técnicas criptográficas aquí descritas son cruciales históricamente y, aplicadas a estructuras como las curvas elípticas, son las que se usan mayoritariamente a día de hoy.

Posteriormente, en el segundo capítulo, se ha visto una introducción a la computación cuántica y expuesto el funcionamiento del algoritmo de Shor, profundizando especialmente en su aplicación para romper la suposición RSA. Este capítulo muestra la importancia de desarrollar criptografía postcuántica, y por tanto justifica el desarrollo de los posteriores capítulos.

Es en estos siguientes capítulos donde se desarrolla la parte de criptografía postcuántica propiamente dicha. En primer lugar, he aprendido lo que son los retículos, los diferentes problemas que se pueden plantear sobre ellos y cómo usarlos para construir criptosistemas y esquemas de firma digital. Los primeros que se han expuesto tienen algún inconveniente de eficiencia o de seguridad, y esto ha motivado la aparición, y justificado la importancia, de los

esquemas Kyber y Dilithium, cuya exposición es el objetivo último de este trabajo. Resulta interesante comparar los diferentes criptosistemas que se han expuesto o mencionado en esta parte del trabajo, lo que se hace en la tabla 5.2. Como vemos, es Kyber el que reúne las mejores propiedades. Algo análogo se podría hacer para los esquemas de firma digital vistos, y de nuevo obtendríamos que Dilithium es el mejor esquema de firma digital expuesto en este trabajo.

Criptosistema	Sobre retículos con estructura	Posee prueba de seguridad	Eficiencia
GGH	NO	NO	NO
Basado en LWE	SÍ	SÍ	Depende de los parámetros
NTRU	SÍ	NO	SÍ
Kyber	SÍ	SÍ	SÍ

Cuadro 5.2: Criptosistemas postcuánticos mencionados en la memoria

El trabajo hecho en este TFG se podría continuar de distintas formas. Para empezar, se podría profundizar más en algunas nociones generales de criptografía, en especial detallar el funcionamiento de algún cifrado simétrico. De este modo, quedaría mejor ilustrado como funcionan los criptosistemas de clave pública CCA seguros en cuya construcción aparecen estos cifrados.

Respecto a la parte de computación cuántica, las opciones de temas en los que profundizar son enormes. En primer lugar, y especialmente de cara a tratar la implementación del algoritmo de Shor en un ordenador cuántico real, sería útil ver cómo este algoritmo se puede implementar utilizando únicamente puertas de uno y dos Qbits. En concreto, sería interesante estudiar cómo implementar de manera eficiente las transformaciones unitarias $\mathbf{U}_f$ utilizadas. Por último, se podría profundizar en la transformada cuántica de Fourier, especialmente explicar su funcionamiento sobre $\mathbb{Z}_N$ con N cualquier número natural y no sólo una potencia de 2.

Para la parte de criptografía postcuántica, falta en este TFG describir cómo se utilizan los problemas sobre retículos vistos para obtener funciones de una vía, siendo esta una buena línea para continuar el trabajo. También, dar las demostraciones de los teoremas que garantizan la seguridad de Kyber y Dilithium. Esta tarea no se ha hecho en este TFG no

solo por falta de tiempo y espacio, sino también por su avanzada complejidad.

En cualquier caso, y a pesar de las líneas de mejora recién expuestas, en el presente trabajo se ha alcanzado el objetivo deseado: motivar la aparición y explicar el funcionamiento de las propuestas de criptografía postcuántica Kyber y Dilithium.

Apéndice A

Criptosistemas de clave privada

Como hemos visto, un criptosistema de clave privada o de clave simétrica es un criptosistema en que Alice y Bob comparten la misma clave. En este apéndice me limitaré a definir lo que es un cifrado simétrico y a dar las nociones básicas de seguridad que serán útiles para desarrollar los criptosistemas de clave pública sobre los que trata este TFG. Las definiciones de este apéndice provienen de [1] y de [15].

Definición A.0.1. Un *cifrado computacional* $\varepsilon = (E, D)$ es un par de algoritmos eficientes E y D. El algoritmo E es el algoritmo de cifrado, que toma una clave k y un mensaje en claro m, y produce un mensaje cifrado c. D es el algoritmo de descifrado, que toma una clave k y un mensaje cifrado c, y devuelve un mensaje m. Si las claves están en un espacio finito $\mathcal{K}$, los mensajes en un espacio finito $\mathcal{M}$, y los mensajes cifrados en un espacio finito $\mathcal{C}$, diremos que ε está definido sobre $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. A menudo, a los cifrados computacionales se les llama simplemente cifrados. En general permitiremos que el algoritmo E sea probabilístico, aunque esto no es necesario como en el caso de criptosistemas de clave pública.

Definición A.0.2 (Propiedad de corrección). Decimos que un cifrado $\varepsilon = (E, D)$ definido sobre $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ tiene la propiedad de corrección si para todo $k \in \mathcal{K}$ y para todo $m \in \mathcal{M}$,

$$Pr[D(k, E(k, m)) = m] = 1$$

A continuación, daré algunas definiciones para la seguridad de los cifrados simétricos. Para nuestro objetivo, bastará definir las nociones de seguridad semántica y CPA, así como las nociones de CCA y 1CCA seguridad. Formalizaremos las diferentes nociones de seguridad mediante juegos:

Juego A.0.1 (CPA seguridad para cifrados simétricos). Dado un cifrado simétrico $\varepsilon = (E, D)$ definido sobre $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ y dado un adversario $\mathcal{A}$, definimos dos experimentos. Para $b = 0, 1$, definimos el experimento b como:

- El retador escoge aleatoriamente una clave $k \in \mathcal{K}$.
- $\mathcal{A}$ le hace al retador una serie de preguntas: Para $i = 1, 2, \ldots,$ la pregunta $i-$ésima consiste en un par $(m_{i0}, m_{i1}) \in \mathcal{M}^2$. El retador calcula $c_i \leftarrow E(k, m_{ib})$ y se lo manda a $\mathcal{A}$.
- Al final del juego, el adversario elige un bit $\hat{b} \in \{0, 1\}$.

Si W_b es el suceso en que $\mathcal{A}$ saca un 1 en el experimento b, definimos la ventaja de $\mathcal{A}$ respecto a ε como

$$\text{CPAadv}[\mathcal{A}, \varepsilon] := |Pr[W_0] - Pr[W_1]|$$

Definición A.0.3 (Cifrado simétrico CPA seguro). Un cifrado ε se dice que es semánticamente seguro frente a ataques de texto plano elegido, o simplemente CPA seguro, si para todos los adversario eficientes $\mathcal{A}$ la cantidad $\text{CPAadv}[\mathcal{A}, \varepsilon]$ es despreciable.

Juego A.0.2 (CCA seguridad para cifrados simétricos). Dado un cifrado simétrico $\varepsilon = (E, D)$ definido sobre $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ y dado un adversario $\mathcal{A}$, definimos dos experimentos. Para $b = 0, 1$, definimos el experimento b como:

- El retador escoge aleatoriamente una clave $k \in \mathcal{K}$.
- $\mathcal{A}$ le hace al retador una serie de preguntas, que pueden ser de dos tipos:
  - *Preguntas de cifrado*: para $i = 1, 2, \ldots,$ la pregunta de cifrado $i-$ésima consiste en un par $(m_{i0}, m_{i1}) \in \mathcal{M}^2$. El retador calcula $c_i \leftarrow E(k, m_{ib})$ y se lo manda a $\mathcal{A}$.
  - *Preguntas de descifrado*: para $i = 1, 2, \ldots,$ la pregunta de descifrado $j-$ésima consiste en un $\hat{c}_j \in \mathcal{C}$ que no está entre las respuestas a preguntas de cifrado. El retador calcula $\hat{m}_j = D(k, \hat{c}_j)$ y se lo manda a $\mathcal{A}$
- Al final del juego, el adversario devuelve un bit $\hat{b} \in \{0, 1\}$.

Si W_b es el suceso en que $\mathcal{A}$ saca un 1 en el experimento b, definimos la ventaja de $\mathcal{A}$ respecto a ε como

$$\text{CCAadv}[\mathcal{A}, \varepsilon] := |Pr[W_0] - Pr[W_1]|$$

Definición A.0.4 (Cifrado simétrico CCA seguro). Un cifrado ε se dice que es semánticamente seguro frente a ataques de texto cifrado elegido, o simplemente CCA seguro, si para todos los adversario eficientes $\mathcal{A}$ la cantidad $\mathrm{CCAadv}[\mathcal{A}, \varepsilon]$ es despreciable.

En muchos casos, una clave se usa únicamente para cifrar un mensaje. En estos casos, son utilizadas las siguientes nociones de seguridad:

Definición A.0.5 (Seguridad semántica). Si en el juego de la CPA seguridad el adversario está restringido a hacer una sola pregunta, denotaremos su ventaja como $\mathrm{SSadv}[\mathcal{A}, \varepsilon]$. Diremos que el cifrado ε es seguro semánticamente si para todo adversario eficiente $\mathcal{A}$ el valor $\mathrm{SSadv}[\mathcal{A}, \varepsilon]$ es despreciable.

Definición A.0.6 (Cifrado simétrico 1CCA seguro). Si en el juego de la CCA seguridad el adversario está restringido a hacer una sola pregunta de cifrado, denotaremos su ventaja como $\mathrm{1CCAadv}[\mathcal{A}, \varepsilon]$. Diremos que el cifrado ε es 1CCA seguro si para todo adversario eficiente $\mathcal{A}$ el valor $\mathrm{1CCAadv}[\mathcal{A}, \varepsilon]$ es despreciable.

Bibliografía

[1] Boneh D., Shoup V., *A Graduate Course in Applied Cryptography.*, (2020). Disponible en: https://crypto.stanford.edu/ dabo/cryptobook/BonehShoup_0_5.pdf, Último acceso: 01/05/2023.

[2] Stinson, D.R., *Cryptography: Theory and practice* (3ª ed.), Boca Raton: Chapman & Hall/CRC, (2006).

[3] Fujisaki E., Okamoto T., *Secure Integration of Asymmetric and Symmetric Encryption Schemes*, Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings, 1666, 537-554, Springer, (1999).

[4] Pointcheval D., Stern J., *Security Proofs for Signature Schemes.*, Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding, 1070, 387-398, Springer, (1996).

[5] David Mermin N., *Quantum Computer Science - An Introduction.*, Cambridge University Press, (2007).

[6] Childs A., *Lecture 2: The HSP and Shor's Algorithm for discrete log*, Quantum algorithms (CO 781, 10 de enero de 2008), University of Waterloo, (2008).

[7] Shor P. W., *Algorithms for Quantum Computation: Discrete Logarithms and Factoring*, Proceedings 35th Annual Symposium on Foundations of Computer Science, 124-134, (1994).

[8] Pellet-Mary A. *Lattice-based crypto, part 1: Algorithmic problems over lattices*, Budapest, Summer school in post-quantum crypto-

graphy, (2022). Disponible en https://apelletm.pages.math.cnrs.fr/page-perso/documents/enseignement/summer_school_Budapest/lecture_1.pdf, Último acceso: 01/05/2023.

[9] Pellet-Mary A., *Lattice-based crypto, part 2: Protocols and structured lattices*, Budapest, Summer school in post-quantum cryptography, (2022). Disponible en https://apelletm.pages.math.cnrs.fr/page-perso/documents/enseignement/summer_school_Budapest/lecture_2.pdf, Último acceso: 01/05/2023.

[10] Pellet-Mary A., *Algebraic lattices for cryptography*, Edinburgh, Fundations and applications of lattice-based cryptography workshop, (2022). Disponible en https://apelletm.pages.math.cnrs.fr/page-perso/documents/presentations/Edinburgh_Alice_part_1.pdf, Último acceso: 01/05/2023.

[11] Micciancio D., Regev O., *Lattice-based Cryptography*, En: Bernstein, D.J., Buchmann, J., Dahmen, E. (eds) Post-Quantum Cryptography. Springer, Berlin, Heidelberg., (2009).

[12] Regev O., *On lattices, learning with errors, random linear codes, and cryptography*, Journal of the ACM, 34:1-34:40, (2009).

[13] Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J. M., Schwabe, P., Seiler, G., & Stehle, D., *CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM.*, 2018 IEEE European Symposium on Security and Privacy (EuroS& P), 353-367, (2018).

[14] Hofheinz D., Hövelmanns K., & Kiltz E., *A modular analysis of the Fujisaki-Okamoto transformation*, IACR Cryptology ePrint Archive report 2017/604, (2017). Disponible en https://eprint.iacr.org/2017/ 604, Último acceso: 01/05/2023.

[15] Cramer R., Shoup V, *Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack*, Cryptology ePrint Archive, Paper 2001/108, (2001). Disponible en https://eprint.iacr.org/2001/108, Último acceso: 01/05/2023.

[16] Ducas L., Lepoint T., Lyubashevsky V., Schwabe P., Seiler G., Stehlé D., *CRYSTALS - Dilithium: Algorithm Specifications and Supporting Documentation (Version 3.1)*, (2021). Disponible en https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf, Último acceso: 01/05/2023.