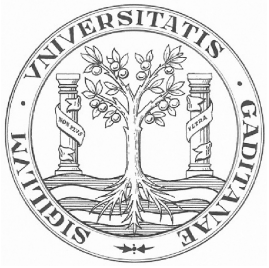


UNIVERSIDAD DE CÁDIZ

FACULTAD DE CIENCIAS

GRADO EN MATEMÁTICAS

CRIPTOGRAFÍA POSCUÁNTICA

Trabajo de fin de grado presentado por

Juan Antonio Guitarte Fernández

Tutor: Dr. Nombre apellidos del tutor

Firma del alumno Firma del tutor

Puerto Real, Cádiz, Julio de 2.020

Abstract

(por hacer)

(por hacer)

Resumen

(por hacer)

Agradecimientos

(por hacer)

Juan Antonio Guitarte Fernández

junio 2020

Índice general

CAPITULO

1

Introducción

(por hacer)

CAPITULO

2

Planteamiento y motivación

2.1 El problema de los ordenadores cuánticos

Hoy día, la criptografía está más presente que nunca en nuestro día a día: hacer compras por Internet, navegar por casi cualquier página web, chatear a través del teléfono móvil... Gracias a la criptografía, podemos mantener nuestras comunicaciones privadas y asegurarnos de que cualquier pago que realicemos o documento que publiquemos sólo podemos hacerlo nosotros, es decir, que nadie pueda falsificarlo.

El continuo desarrollo de los ordenadores cuánticos, que romperán los principales algoritmos de firma digital y criptosistemas de clave pública usados hoy en día (por ejemplo, *RSA*[1], *DSA*[2] y *ECDSA*[3]), puede hacer pensar que cuando la computación cuántica sea una realidad, la criptografía quedará obsoleta, que será imposible modificar información para que sea incomprensible o infalsificable por atacantes y personas no autorizadas; y que por tanto, la única forma de proteger nuestras comunicaciones y nuestros datos será aislarlos físicamente de ellos, por ejemplo, con dispositivos USB cerrados bajo llave en un maletín. Pero, ¿hasta qué punto es esto cierto?

Un estudio más detallado de los algoritmos criptográficos existentes muestra, sin embargo, que existen muchos otros criptosistemas más allá del *RSA*, *DSA* y *ECDSA*:

- **Criptografía basada en funciones hash**. El ejemplo más destacado dentro de este grupo es el sistema de firma con clave pública basado en árboles hash de Merkle

(en inglés, *Merkle's hash-tree public-key signature system*) de 1979, basado en un sistema de firma digital de un solo uso de Lamport y Diffie.

- **Criptografía basada en códigos**. El ejemplo clásico es el sistema de encriptación de clave pública con códigos Goppa ocultos de McEliece (1978).

- **Criptografía basada en retículos**. El ejemplo que más interés ha conseguido atraer, aunque no es el primero propuesto históricamente, esl el sistema de encriptación de clave pública "NTRU" de Hoffstein-Pipher-Silverman (1998).

- **Criptografía de ecuaciones cuadráticas de varias variables**. Uno de los ejemplos más interesantes es el sistema de firma con clave pública "HFE^{v-}" de Patarin (1996), que generaliza una propuesta de Matsumoto e Imai.

- **Criptografía de clave secreta**. El ejemplo más conocido (y usado actualmente) es el cifrado "Rijndael" de Daemen-Rijmen (1998), renombrado como "AES", siglas que significan Estándar de Encriptación Avanzada (Advanced Encryption Standard).

Se cree que todos estos sistemas son resistentes a los ordenadores clásicos y cuánticos, es decir, que no existe un algoritmo eficiente que pueda ser implementado en un ordenador clásico o cuántico que rompa estos sistemas [4]. El algoritmo de Shor (el cual analizaremos más adelante en este trabajo), que permite resolver de manera eficiente el problema de la factorización de números enteros en ordenadores cuánticos (y por tanto rompe los sistemas de criptografía clásica como el *RSA*), no ha podido ser aplicado a ninguno de estos sistemas. Aunque existen otros algoritmos cuánticos, como el algoritmo de Grover, que pueden ser aplicados a algunos de estos sistemas, no son tan eficientes como el algoritmo de Shor y los criptógrafos pueden compensarlo eligiendo claves un poco más grandes ([5], [6]).

Hay que notar que esto no implica que estos sistemas sean totalmente seguros. Este es un problema muy común en criptografía: algunas veces se encuentran ataques a sistemas que son devastadores, demostrando que un sistema es inútil para la criptografía; otras veces, se encuentran ataques que no son tan devastadores pero que obligan a elegir claves más grandes para que sigan siendo seguros; y otras, se estudian criptosistemas durante años sin encontrar ningún ataque efectivo. En este punto, la comunidad puede ganar confianza en el sistema creyendo que el mejor ataque posible ya ha sido encontrado, o que existe muy poco margen de mejora.

2.2 Criptosistemas

El objetivo principal de la criptografía es permitir que dos personas, normalmente referidas como Alice y Bob, puedan comunicarse entre ellas a través de un canal inseguro de tal manera que una tercera persona, Oscar, no pueda entender qué están diciendo entre ellos, aun teniendo acceso a toda la conversación. La información que Alice quiere enviar a Bob la denominamos "texto plano", aunque no tiene que ser necesariamente texto; puede tener la estructura que deseemos: datos numéricos, cadenas de bits, sonido... Alice encripta el texto plano usando una "clave" que solo conocen Alice y Bob, obteniendo así un "texto encriptado". Oscar, al ver la información a través del canal inseguro, no puede determinar cuál era el texto plano original; pero Bob, que sí conoce la clave, puede desencriptar el texto cifrado y recuperar el texto plano.

Formalmente, un criptosistema se define de la siguiente manera:

Definición 2.1. Un *criptosistema* es una 5-tupla $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ que satisface las siguientes condiciones:

1. $\mathcal{P}$ es un conjunto finito de *textos planos* posibles,
2. $\mathcal{C}$ es un conjunto finito de *textos cifrados* posibles,
3. $\mathcal{K}$ es el conjunto finito de todas las claves posibles,
4. Para cada $K \in \mathcal{K}$, existen dos aplicaciones $e_K : \mathcal{P} \to \mathcal{C}$ y $d_K : \mathcal{C} \to \mathcal{P}$, denominadas *regla de encriptación* y *regla de desencriptación* respectivamente, que verifican que $d_K(e_K(x)) = x$ para todo $x \in \mathcal{P}$.

La propiedad 4, que es la más importante, asegura que conociendo la clave $K \in \mathcal{K}$, se puede recuperar el texto sin cifrar original usando la función d_K. El proceso por el cual Alice y Bob utilizarían un criptosistema es el siguiente:

1. Alice y Bob seleccionan una misma clave $K \in \mathcal{K}$ de forma aleatoria.
2. Supongamos que Alice quiere enviar un mensaje $x = x_1 x_2 \cdots x_n$, con $x_i \in \mathcal{P}$ para todo $1 \leq i \leq n$. Alice calcula, para cada $1 \leq i \leq n$, $y_i = e_K(x_i)$, resultando en el mensaje cifrado
$$y = y_1 y_2 \cdots y_n$$

que Alice envía a través del canal inseguro a Bob.

3. Bob, al recibir y, calcula usando la clave K que conoce $d_K(y_i)$, que coincidirán con los x_i originales por la propiedad 4 de la Definición 2.1, obteniendo así el texto original x.

Hay que notar que para que este método funcione, Alice y Bob deben escoger la misma clave K para encriptar y desencriptar los mensajes. En algunos criptosistemas (como el AES mencionado anteriormente), sabiendo e_K o d_K, es sencillo obtener la otra función porque se conoce la clave secreta K. Un criptosistema de este tipo se denomina *criptosistema de clave simétrica*, ya que si un atacante obtuviese la función e_K o d_K, podría romper el sistema desencriptando los mensajes cifrados, bien usando d_K directamente en el segundo caso o bien calculando d_K a partir de e_K a través de la clave en el primero.

Por tanto, es fundamental que Alice y Bob, antes de iniciar cualquier comunicación a través del canal inseguro, se pongan de acuerdo a través de un canal seguro en la clave que van a utilizar. En la práctica, esto es muy difícil de conseguir (por ejemplo, en el caso de Internet). Para resolver este problema, existen los *criptosistemas de clave pública*.

La idea tras estos criptosistemas es que dada una función de encriptación e_K, sea computacionalmente infactible calcular d_K. En este caso, el receptor del mensaje, Bob, publicaría una *clave pública* que permitiría a cualquier persona determinar una función de encriptación e_K. Así, Alice encriptaría el mensaje que quiere enviar usando esta función. El mensaje cifrado llegaría entonces a Bob, que es el único que conoce su *clave privada* con la cual puede calcular la función de desencriptación d_K correspondiente a e_K, desencriptando así el mensaje.

Estos criptosistemas son los que se ven principalmente afectados por la aparición de los ordenadores cuánticos: mientras que en un ordenador clásico puede ser muy difícil calcular la clave privada a partir de la clave pública, pueden existir algoritmos cuánticos que resuelvan el problema en un tiempo razonable. Es por ello que se necesitan nuevos sistemas en los que no existan algoritmos conocidos, ni clásicos ni cuánticos, que permitan calcular eficientemente d_K a partir de e_K.

2.3 Sistemas de firma

El otro gran objetivo de la criptografía es permitir la firma de documentos. En este caso, Alice publicaría el mensaje o documento con una *firma* que permite a cualquier persona

verificar que el mensaje sólo ha podido ser escrito por Alice. De esta manera, un atacante Oscar que quisiese publicar un documento haciéndose pasar por Alice, debe generar una firma con él para que pueda ser validado por el resto de personas. El proceso de firmar, por tanto, debe ser computacionalmente sencillo para Alice, pero infactible para Oscar, como sucede en los criptosistemas de clave pública.

Formalmente, un sistema de firma se define de la siguiente manera:

Definición 2.2. Un *sistema de firma* es una 5-tupla $(\mathcal{P}, \mathcal{A}, \mathcal{K}, \mathcal{S}, \mathcal{V})$ que verifica:

1. $\mathcal{P}$ es un conjunto finito de posibles *mensajes*,
2. $\mathcal{A}$ es un conjunto finito de *firmas* posibles,
3. $\mathcal{K}$ es el conjunto de las claves posibles,
4. Para cada $K \in \mathcal{K}$, hay dos aplicaciones $sig_K \in S$ y $ver_K \in V$, denominadas algoritmos de *firma* y *verificación* respectivamente, siendo $sig_K : \mathcal{P} \to \mathcal{A}$ y $ver_K : \mathcal{P} \times \mathcal{A} \to \{0, 1\}$, que verifican para cada mensaje $x \in \mathcal{P}$ y cada firma $y \in \mathcal{A}$:
$$ver_K(x, y) = \begin{cases} 1 & \text{si } y = sig_K(x) \\ 0 & \text{si } y \neq sig_K(x) \end{cases}$$
A un par ordenado de la forma $(x, y) \in \mathcal{P} \times \mathcal{A}$ se le denomina *mensaje firmado*.

CAPITULO

3

Espacios de Hilbert

3.1 Definiciones básicas

Un concepto vital en el desarrollo de la teoría de ordenadores cuánticos es el concepto de espacio de Hilbert, que se apoya en los productos escalares. En este desarrollo, consideraremos en todo momento que el espacio vectorial subyacente X es de dimensión finita, es decir, $\dim X = n \in \mathbb{N}$; ya que no trabajaremos en los capítulos siguientes con espacios de Hilbert de dimensión infinita. Esto simplificará algunas demostraciones aquí presentadas; no obstante, con un poco más de esfuerzo, se pueden hacer para el caso general.

Definición 3.1. Sean E y F dos espacios vectoriales sobre un cuerpo $\mathbb{K}$ (consideraremos $\mathbb{R}$ ó $\mathbb{C}$). Una aplicación $u : E \to F$ es una *aplicación semilineal* si para cada $x, y \in E$ y para cada $\alpha \in \mathbb{K}$ verifica:

1. $u(x + y) = u(x) + u(y)$
2. $u(\alpha x) = \overline{\alpha} u(x)$

donde $\bar{\cdot}$ denota la conjugación compleja.

Esta noción generaliza el concepto de aplicación lineal en espacios vectoriales reales, puesto que si $\mathbb{K} = \mathbb{R}$, entonces el concepto de aplicación semilineal coincide con el de

lineal (ya que $\overline{\alpha} = \alpha$ para todo $\alpha \in \mathbb{R}$). El siguiente es una generalización del concepto de forma bilineal.

Definición 3.2. Sea E un espacio vectorial sobre un cuerpo $\mathbb{K}$. Una aplicación $B : E \times E \to \mathbb{K}$ se dice que es una *forma sesquilineal* si es lineal respecto de la primera componente y semilineal respecto de la segunda; es decir, si para cada $x, x', y, y' \in E$ y para cada $\alpha, \lambda \in \mathbb{K}$ se verifica:

1. $B(x + x', y) = B(x, y) + B(x', y)$
2. $B(\lambda x, y) = \lambda B(x, y)$
3. $B(x, y + y') = B(x, y) + B(x, y')$
4. $B(x, \alpha y) = \overline{\alpha} B(x, y)$

Si además B verifica que $B(x, y) = \overline{B(y, x)}$, entonces se dice que es *hermítica*.

Las *aplicaciones sesquilineales* entre dos espacios vectoriales se definen de forma análoga con las propiedades 1-4 de la definición anterior, cambiando el producto en $\mathbb{K}$ por el producto exterior del espacio vectorial de llegada.

Definición 3.3. Sea E un espacio vectorial y sea B una forma sesquilineal y hermítica sobre E. Si para cada $x \in E$ se verifica $B(x, x) \geq 0$ entonces se dirá que B es *positiva*.

Si además se verifica que $x = 0$ cuando $B(x, x) = 0$ entonces se dice que B es *definida positiva*.

Ya estamos en condiciones de presentar lo que es un espacio de Hilbert.

Definición 3.4. Sea X un espacio vectorial sobre $\mathbb{K}$. Un *producto escalar* o *producto interior* en X es una forma sesquilineal hermítica definida positiva B sobre $X \times X$. Se suele denotar por $(x|y) = B(x, y)$ o bien por $\langle x, y \rangle = B(x, y)$.

Un espacio vectorial X que está dotado de un producto escalar diremos es un *espacio prehilbertiano*.

Un espacio prehilbertiano es un *espacio de Hilbert* si es completo.

Esta definición generaliza el concepto de producto escalar en espacios vectoriales reales.

Ejemplo 3.1. Sea $X = \mathbb{C}^2$. El producto escalar usual en este espacio se define como

$$\langle (z_1, z_2), (z_1', z_2') \rangle = z_1\overline{z_1'} + z_2\overline{z_2'}$$

En efecto, es un producto escalar, ya que:

1. $\langle (z_1, z_2)+(z_3, z_4), (z_1', z_2') \rangle = (z_1+z_3)\overline{z_1'}+(z_2+z_3)\overline{z_2'} = z_1\overline{z_1'}+z_2\overline{z_2'}+z_3\overline{z_1'}+z_4\overline{z_2'} = \langle (z_1, z_2), (z_1', z_2') \rangle + \langle (z_3, z_4), (z_1', z_2') \rangle$.
2. $\langle \lambda(z_1, z_2), (z_1', z_2') \rangle = \lambda z_1\overline{z_1'} + \lambda z_2\overline{z_2'} = \lambda\langle (z_1, z_2), (z_1', z_2') \rangle$
3. $\langle (z_1, z_2), (z_1', z_2') + (z_3', z_4') \rangle = z_1\overline{z_1' + z_3'} + z_2\overline{z_2' + z_4'} = z_1\overline{z_1'} + z_2\overline{z_2'} + z_1\overline{z_3'} + +z_2\overline{z_4'} = \langle (z_1, z_2), (z_1', z_2') \rangle + \langle (z_1, z_2), (z_3', z_4') \rangle$.
4. $\langle (z_1, z_2), \alpha(z_1', z_2') \rangle = \lambda z_1\overline{\alpha z_1'} + z_2\overline{\alpha z_2'} = \overline{\alpha}\langle (z_1, z_2), (z_1', z_2') \rangle$

Análogamente al caso real, dado $H \subseteq X$ un subespacio vectorial de un espacio de Hilbert X, podemos definir

$$H^{\perp} = \{x \in X : \langle x, h \rangle = 0 \text{ para todo } x \in X\}$$

Y es sencillo comprobar, por las propiedades del producto escalar, que es un subespacio vectorial y que $\dim H + \dim H^{\perp} = \dim X$.

También podemos definir el concepto de base ortonormal como una base $\{u_i\}_{i=1,\cdots,n}$ que verifica que $\langle u_i, u_j \rangle = \delta_{ij}$ (la delta de Kronecker). Análogamente al caso real, todo espacio de Hilbert tiene una base ortonormal; pues siempre podemos construir una usando el método de Gram-Schmidt (ver [7], pág. 167).

Notemos que en un espacio de Hilbert X es posible definir una norma de la siguiente manera: para cada $x \in X$,

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

Es fácil ver que esta norma está bien definida, por ser $\langle x, x \rangle \geq 0$ (es definida positiva), y que por las propiedades del producto escalar, es una norma. De esta manera, todo espacio de Hilbert es un espacio normado, y al ser completo, es un *espacio de Banach* (ver [7], pág. 27). Este hecho justifica el exigir la propiedad de completitud a un espacio de Hilbert, pues esta norma induce una métrica en X, y gracias a ello podemos hablar de sucesiones de Cauchy y sucesiones convergentes (un conjunto es completo si toda sucesión de Cauchy en él es convergente).

Ejemplo 3.2. Sea $X = \mathbb{C}^2$. La *norma usual* inducida por el producto escalar usual es

$$\|(z_1, z_2)\| = \sqrt{\langle (z_1, z_2), (z_1, z_2) \rangle} = \sqrt{z_1\overline{z_1} + z_2\overline{z_2}} = \sqrt{|z_1|^2 + |z_2|^2}$$

Una propiedad conocida es la desigualdad de Cauchy-Schwarz, que se verifica en espacios prehilbertianos.

Teorema 3.1 (Desigualdad de Cauchy-Schwarz). *Sea E un espacio vectorial y $B(x, y)$ una forma sesquilineal hermítica y positiva sobre E. Se verifica, para cada $x, y \in E$, que*

$$|B(x, y)| \leq B(x, x)^{1/2} B(y, y)^{1/2}$$

Una demostración clásica de este hecho puede consultarse en [7], pág. 154.

Como consecuencia inmediata de la definición de $\|\cdot\|$, tenemos el siguiente importante resultado.

Corolario 3.1. *Sea X un espacio prehilbertiano. Entonces, para cada $x, y \in X$, se verifica que:*

$$|\langle x, y \rangle| \leq \|x\| \|y\|$$

3.2 El operador adjunto y operadores unitarios

Ya que los espacios de Hilbert son espacios vectoriales, podemos hablar de aplicaciones lineales entre ellos y de formas lineales. De entre ellas, en el contexto de la computación cuántica nos interesarán aquellas que sean *unitarias*. Para poder definirlas correctamente, necesitamos un poco de teoría de espacios de Hilbert.

Lema 3.1. *Sean X, Y espacios de Hilbert y $f : X \to Y$ una aplicación lineal. Entonces, f es continua si y sólo si $f(S_X) = f(\{x \in X : \|x\| = 1\})$ es acotado.*

Una demostración de este hecho puede consultarse en [7].

Lema 3.2. *Sean X, Y, Z tres espacios normados y $u : X \times Y \to Z$ una aplicación sesquilineal, entonces las siguientes afirmaciones son equivalentes:*

i) u es continua en $(0, 0)$.

ii) Existe $M > 0$ tal que $\|u(x, y)\| \leq M\|x\|\|y\|$ para cada $(x, y) \in X \times Y$

iii) u es continua en $X \times Y$.

Demostración.

iii) $\implies$ i) Es trivial.

i) $\implies$ ii) Tomando $\epsilon = 1$, entonces por i) existe $\delta > 0$ tal que si $\|x\| \leq \delta$, $\|y\| \leq \delta$, entonces $\|u(x, y)\| \leq \epsilon = 1$. Si fuese $x = 0$ ó $y = 0$, el resultado es claro puesto que

$$\|u(0, y)\| = \|u(x, 0)\| = \|0\| = 0 \leq M\|x\|\|0\| = M\|0\|\|y\| = 0$$

Si es $x \neq 0, y \neq 0$, entonces se verifica que $\left\| \frac{\delta x}{\|x\|} \right\| = \frac{\delta\|x\|}{\|x\|} = \delta$, $\left\| \frac{\delta y}{\|y\|} \right\| = \frac{\delta\|y\|}{\|y\|} = \delta$, y por tanto por el razonamiento anterior,

$$\left\| u\left(\frac{\delta x}{\|x\|}, \frac{\delta y}{\|y\|} \right) \right\| \leq 1$$

Aplicando finalmente sesquilinealidad, tenemos que

$$\frac{\delta^2}{\|x\|\|y\|}\|u(x, y)\| \leq 1$$

$$\|u(x, y)\| \leq \delta^2\|x\|\|y\|$$

ii) $\implies$ iii) Sea $(a, b) \in X \times Y$. Se verifican, para cada $(x, y) \in X \times Y$, usando la desigualdad triangular y la sesquilinealidad:

$$\|u(x, y) - u(a, b)\| = \|u(x, y) - u(a, y) + u(a, y) - u(a, b)\| \leq \|u(x - a, y)\| + \|u(a, y - b)\|$$

Y además, por hipótesis, se verifica que

$$\|u(x, y) - u(a, b)\| \leq \|u(x - a, y)\| + \|u(a, y - b)\| \leq M\|x - a\|\|y\| + M\|a\|\|y - b\|$$

Dado $\varepsilon > 0$, tomando $\delta_2 > 0$ de modo que $\delta_2 < \frac{\varepsilon}{2M\|a\|}$ si $a \neq 0$ y $\delta_2 < \frac{\varepsilon}{2M(\|b\|+\delta_2)}$, tenemos que si $\|x - a\| < \delta_1$ y $\|y - b\| < \delta_2$, entonces $\|u(x, y) - u(a, b)\| < \varepsilon$, lo que queríamos probar. Si fuese $a = 0$, entonces el resultado es trivial tomando $\delta_1 = \frac{\varepsilon}{\|y\|}$.

□

Teorema 3.2 (Fréchet-Riesz). *Sea X un espacio de Hilbert y sea $f : X \to \mathbb{K}$ una aplicación lineal y continua. Existe un único $a \in X$ tal que $f = f_a$, donde $f_a : X \to \mathbb{K}$ es la aplicación definida por $f_a(x) = \langle x, a\rangle$.*

Demostración. Sea $H = \ker f$. Observemos que se verifica, por ser f lineal:

$$\dim H + \dim \text{Im } f = \dim X$$

Como $\dim \mathbb{K} = 1$ como $\mathbb{K}$-espacio vectorial, entonces $\dim \text{Im } f \leq 1$. Si fuese 0, entonces es $f = 0$ (la aplicación nula), y bastaría tomar $a = 0$.

Supongamos pues, que $\dim \text{Im } f = 1$. Entonces, $\dim H = \dim X - 1$, y por ser $\dim H + \dim H^\perp = \dim X$, entonces $\dim H^\perp = 1$. Luego podemos tomar $b \in H^\perp$ con $b \neq 0$, y se cumple que $H^\perp = \mathcal{L}(b)$. De esta manera, como $X = H + H^\perp$, cada $x \in X$ se puede expresar de la forma $x = y + \alpha b$, $y \in H, \alpha \in \mathbb{K}$.

Tomamos $a = \frac{\overline{f(b)}}{\|b\|^2} b$. Veamos que a verifica las propiedades del enunciado: sea $x \in X$. Por un lado,

$$f(x) = f(y + \alpha b) = f(y) + \alpha f(b) = \alpha f(b)$$

por la linealidad de f y que $y \in \ker f$.

Por otro lado,

$$\langle x, a\rangle = \langle y + \alpha b, \frac{\overline{f(b)}}{\|b\|^2} b\rangle = \frac{f(b)}{\|b\|^2}\langle y, b\rangle + \alpha \frac{f(b)}{\|b\|^2}\langle b, b\rangle = \alpha f(b)$$

Ya que $\langle y, b\rangle = 0$ por ser $y \in H$, $b \in H^\perp$. Esto prueba que $f = f_a$.

Veamos la unicidad. Sea $a' \in X$ tal que $f = f_{a'}$. Entonces, para cada $x \in X$, se verifica que:

$$f(x) = \langle x, a\rangle = \langle x, a'\rangle$$

de donde deducimos que

$$0 = \langle x, a - a'\rangle$$

Lo que significa que $a - a' \in X^\perp = \{0\}$. De aquí, $a = a'$. □

El siguiente teorema finalmente nos termina de preparar el camino para definir los operadores unitarios.

Teorema 3.3. *Sea X un espacio de Hilbert y sea $B : X \times X \to \mathbb{K}$ una forma sesquilineal y continua. Entonces, existe una única aplicación lineal y continua $f : X \to X$ tal que $B(x, y) = \langle x, f(y)\rangle$ para cada $(x, y) \in X \times X$.*

Demostración. Fijado $y \in X$, podemos definir $g_y : X \to \mathbb{K}$ dada por $g_y(x) = B(x, y)$. Claramente g_y es lineal por serlo la primera componente de B, y además tenemos que, gracias al Lema 3.2, por ser B continua:

$$|g_y(x)| = |B(x, y)| \leq M\|x\|\|y\|$$

Esto implica que, para $(x, y) \in S_X$, $|g_y(x)| \leq M$, y por tanto, g_y es continua. Usando ahora el Teorema 3.2, existe un único $z_y \in X$ tal que, para cada $x \in X$, se verifica que $g_y(x) = \langle x, z_y\rangle$. Esto permite definir una aplicación $f : X \to X$ como $f(y) = z_y$.

Por construcción, f verifica la igualdad del enunciado, ya que

$$\langle x, f(y)\rangle = \langle x, z_y\rangle = g_y(x) = B(x, y)$$

Veamos que f es lineal: sean $y, z \in X$, $\alpha, \beta \in \mathbb{K}$. Entonces, aplicando la sesquilinealidad de B, se tiene que

$$\begin{aligned}\langle x, f(\alpha y + \beta z)\rangle &= B(x, \alpha y + \beta z) = \\ &= \overline{\alpha}B(x, y) + \overline{\beta}B(x, z) = \overline{\alpha}\langle x, f(y)\rangle + \overline{\beta}\langle x, f(z) = \langle x, \alpha f(y) + \beta f(z)\rangle\end{aligned}$$

Con lo que tenemos que, para cada $x \in X$,

$$0 = \langle x, f(\alpha y + \beta z) - (\alpha f(y) + \beta f(z))\rangle$$

Y por tanto $f(\alpha y + \beta z) - (\alpha f(y) + \beta f(z)) \in X^{\perp} = \{0\}$.

Además, f es continua. Ya que para cada $x, y \in X$ es $|\langle x, f(y)\rangle| = |B(x, y)| \leq M\|x\|\|y\|$ (por el Lema 3.2). En particular, para $x = f(y)$, tenemos que

$$\|f(y)\|^2 \leq M\|f(y)\|\|y\|$$

De aquí podemos deducir que

$$\|f(y)\| \leq M\|y\|$$

(desigualdad que también se verifica trivialmente si $f(y) = 0$). Si fijamos $y \in S_x$, $\|y\| = 1$, lo que significa que $\|f(y)\| \leq M$ y por tanto f es continua. □

Este teorema nos permite dar una definición importante. Sea X un espacio de Hilbert y sea $A : X \to X$ una aplicación lineal y continua. La aplicación que envía $(x, y) \to \langle A(x), y\rangle$ es sesquilineal y continua[1]. Por tanto, usando el teorema anterior, debe existir una aplicación lineal y continua $A' : X \to X$ de manera que

$$\langle A(x), y\rangle = \langle x, A'(y)\rangle$$

Definición 3.5. Sea X un espacio de Hilbert y sea $A : X \to X$ una aplicación lineal y continua. El *adjunto de* A, denotado A', es la única aplicación lineal y continua $A' : X \to X$ que satisface para cada $x, y \in X$ que

$$\langle A(x), y\rangle = \langle x, A'(y)\rangle$$

Supongamos ahora que $\{u_i\}_{i=1,\cdots,n}$ es una base ortonormal de X. Sea ahora $(k_{ij})_{i,j=1,\cdots,n}$ la matriz asociada de A y $(k'_{ij})_{i,j=1,\cdots,n}$ la de A' . Se verifica que, por ser la base ortonormal:

$$\langle A(u_j), u_i\rangle = \left\langle \sum_{l=1}^{n} k_{lj}u_l, u_i \right\rangle = \sum_{l=1}^{n} k_{lj}\langle u_l, u_i\rangle = k_{ij}$$

Entonces, usando que el producto escalar es hermítico:

$$k_{ij} = \langle A(u_j), u_i\rangle = \langle u_j, A'(u_i)\rangle = \overline{\langle A'(u_i), u_j\rangle} = \overline{k'_{ji}}$$

Es decir, tomando conjugados e intercambiando los papeles de i y de j:

$$k'_{ij} = \overline{k_{ji}}$$

para cada $i, j = 1, \cdots, n$. Hemos probado:

Proposición 3.1. *Sea X un espacio de Hilbert y sea $A : X \to X$ una aplicación lineal y continua. Sea $\mathcal{A}, \mathcal{B}$ las matrices asociadas a A y A' respectivamente fijando una base ortonormal en X. Entonces, se verifica que:*

$$\mathcal{B} = \mathcal{A}^\dagger$$

Donde $\mathcal{A}^\dagger$ denota la matriz traspuesta conjugada de A.

[1]En efecto, $\langle A(x+x'), y\rangle = \langle A(x)+A(x'), y\rangle = \langle A(x), y\rangle + \langle A(x'), y\rangle$ y $\langle A(\alpha x), y\rangle = \langle \alpha A(x), y\rangle = \alpha\langle A(x), y\rangle$ y en la segunda componente es trivial por la sesquilinealidad del producto escalar. La continuidad se debe a que se trata de la composición de dos funciones continuas, A y el producto escalar.

Ejemplo 3.3. Tomemos $X = \mathbb{C}^2$ con la base ortonormal canónica $\{(1,0),(0,1)\}$ y el producto escalar usual dado por $\langle (z_1,z_2),(z_1',z_2')\rangle = z_1\overline{z_1'} + z_2\overline{z_2'}$. Consideramos la aplicación $f : \mathbb{C}^2 \to \mathbb{C}^2$ dada por

$$f(z_1,z_2) = \frac{1}{\sqrt{3}}\left(z_1 + (-1+i)z_2, (1+i)z_1 + z_2\right)$$

Es fácil ver que f es lineal, y que su matriz asociada es

$$A = \frac{1}{\sqrt{3}}\begin{pmatrix} 1 & -1+i \\ 1+i & 1 \end{pmatrix}$$

Entonces, la matriz asociada del adjunto de f viene dada por

$$A' = \frac{1}{\sqrt{3}}\begin{pmatrix} 1 & 1-i \\ -1-i & 1 \end{pmatrix}$$

Es decir, el adjunto de f viene dado por

$$f'(z_1,z_2) = \frac{1}{\sqrt{3}}\left(z_1 + (1-i)z_2, (-1-i)z_1 + z_2\right)$$

Definición 3.6. Sea X un espacio de Hilbert y sea $A : X \to X$ una aplicación lineal y continua. Se dice que A es *unitaria* si $A' \circ A = A \circ A' = Id$, donde $Id : X \to X$ es la aplicación identidad en X.

Según lo visto antes, esta condición puede traducirse matricialmente a que $AA^\dagger = A^\dagger A = I$.

Ejemplo 3.4. El operador definido en el Ejemplo 3.3 es unitario, ya que

$$A'A = \frac{1}{3}\begin{pmatrix} 3 & 0 \\ 0 & 3 \end{pmatrix} = I$$

La siguiente importante propiedad culmina nuestro estudio.

Proposición 3.2. *Sea X un espacio de Hilbert y sea $A : X \to X$ una aplicación lineal y continua. Si A es unitaria, entonces para todo $x, y \in X$, $\langle A(x), A(y)\rangle = \langle x, y\rangle$. En particular, $\|A(x)\| = \|x\|$ para todo $x \in X$.*

Demostración. Para todo $x, y \in X$, $\langle A(x), A(y)\rangle = \langle x, A'(A(y))\rangle$ por definición de operador adjunto, y como es unitario, $A'A = Id$ y tenemos que $\langle A(x), A(y)\rangle = \langle x, y\rangle$.

El segundo resultado se desprende de la definición de norma: $\|A(x)\| = \sqrt{\langle A(x), A(x)\rangle} = \sqrt{\langle x, x\rangle} = \|x\|$. □

Con un poco más de trabajo, es posible ver que el recíproco también es cierto. Por tanto, podemos decir que los únicos operadores que conservan la norma son los unitarios; por eso nos serán de interés en la computación cuántica (como veremos más adelante).

CAPITULO

4

Los algoritmos de Shor y Grover

El objetivo de este capítulo es describir los dos principales algoritmos que ponen en peligro la criptografía clásica que se usa actualmente en gran medida: el algoritmo de Shor, que permite factorizar cualquier número natural en $\mathcal{O}((\log N)^3)$ pasos ([8], pág. 233), donde N es el número a factorizar; y el algoritmo de Grover, que resuelve el problema de la búsqueda: esto es, encontrar dada una función $f : S \to \{0, 1\}$ con $|S| = N \in \mathbb{N}$, un elemento $x \in S$ tal que $f(x) = 1$ en $\mathcal{O}(\sqrt{N})$ operaciones ([8]).

Es importante ver que el primer algoritmo ofrece una reducción de tiempo exponencial respecto de los algoritmos de factorización clásicos. Los algoritmos de factorización más rápidos para ordenadores clásicos usan $\mathcal{O}\left(2^{(k+o(1))(\log N)^{1/3}(\log\log N)^{2/3}}\right)$ operaciones [4]. Sin embargo, en un ordenador clásico el problema de la búsqueda se puede resolver en $\mathcal{O}(N)$ operaciones (evaluar cada elemento de S). Por tanto, al tratarse de una reducción de tiempo cuadrática, no es tan efectivo como el algoritmo de Shor y sus efectos se pueden compensar aumentando el tamaño de las claves que se usan en los sistemas.

4.1 Introducción al algoritmo de Shor

El algoritmo de Shor contiene dos partes, una parte clásica que se ejecutaría en un ordenador clásico, y una parte cuántica que se ejecutaría en un ordenador cuántico. La idea clave del algoritmo subyace en el siguiente resultado.

Proposición 4.1. *Sea N un número compuesto de L bits, y sea $x \in \mathbb{Z}_N$ con $1 < x < N-1$ una solución de la ecuación*

$$x^2 = 1 \mod N \tag{4.1}$$

Entonces, o bien $\mathrm{mcd}(x-1, N)$ o bien $\mathrm{mcd}(x+1, N)$ es un factor no trivial de N y se puede calcular en $O(L^3)$ operaciones.

Demostración. Ya que $x^2 - 1 = 0 \mod N$, entonces $N|_{(x^2+1)=(x+1)(x-1)}$. De aquí, N debe tener un factor común con $(x+1)$ ó con $(x-1)$, es decir, $\mathrm{mcd}(x+1, N) > 1$ ó $\mathrm{mcd}(x-1, N) > 1$. Además, ya que $1 < x < N-1$, entonces $2 < x+1 < N$ y $0 < x-1 < N-2$, en cualquier caso, $x-1 < N$ y $x+1 < N$ y por tanto $\mathrm{mcd}(x-1, N) < N$ y $\mathrm{mcd}(x+1, N) < N$. Esto prueba que ninguno puede ser un factor trivial de N. Mediante el algoritmo de Euclides, estos factores pueden calcularse en $O(L^3)$ operaciones (ver [8], pág. 629). □

En general, encontrar una solución de (4.1) es difícil. Sin embargo, existe una estrategia para abordar este problema. Si $1 < y < N-1$ es cualquier número coprimo con N y resulta que el orden r del elemento y dentro del grupo multiplicativo $(\mathbb{Z}/N\mathbb{Z})^*$ (es decir, el menor natural tal que $y^r = 1 \mod N$) es par, entonces $y^{r/2}$ sería una solución de (4.1); y además cumpliría las hipótesis de la Proposición 4.1 si $y^{r/2} \neq -1 \mod N$ (observemos que no puede ser 1, ya que ello contradiría la definición de r). Resulta que si escogemos este número y aleatoriamente entre 2 y $N-2$ (si no es coprimo con N, entonces habríamos encontrado un factor calculando $\mathrm{mcd}(y, N)$), entonces es muy probable que verifique las condiciones expuestas, lo cual nos permitiría calcular los factores como enuncia la Proposición 4.1. Este hecho se recoge en el siguiente teorema.

Teorema 4.1. *Sea $N = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_m^{\alpha_m}$ una factorización en números primos de un número impar. Sea x un elemento escogido aleatoriamente de $(\mathbb{Z}/N\mathbb{Z})^*$, y sea r el orden de x módulo N. Entonces,*

$$P[r \text{ es par y que } x^{r/2} \neq -1 \mod N] \geq 1 - \frac{1}{2^m}$$

Para demostrarlo, necesitamos primero un lema previo.

Lema 4.1. *Sea p un número primo diferente de 2 y sea $\alpha \in \mathbb{N}$. Sea 2^d la mayor potencia de 2 que divide a $\varphi(p^\alpha)$. Entonces, con probabilidad de un medio 2^d divide al orden de cualquier elemento de $(\mathbb{Z}/p^\alpha\mathbb{Z})^*$.*

Demostración. Por ser $p \neq 2$, entonces es impar, y por tanto $\varphi(p^\alpha) = p^{\alpha-1}(p-1)$ es par. Esto implica que $d \geq 1$. Ya que el grupo $(\mathbb{Z}/p^\alpha\mathbb{Z})^*$ es cíclico, cualquier elemento se escribirá de la forma g^k, con g el generador del grupo y $1 \leq k \leq \varphi(p^\alpha)$. Sea r el orden de dicho elemento.

- Si es k impar, ya que $\left(g^k\right)^r = g^{kr} = 1 \mod p^\alpha$ entonces por el Teorema de Lagrange debe ser $\phi(p^\alpha)|kr$. De aquí, 2^d divide a kr, y por ser k impar, necesariamente 2^d divide a r, el orden del elemento.

- Si es k par, entonces

$$g^{\frac{k}{2}\varphi(p^\alpha)} = \left(g^{\varphi(p^\alpha)}\right)^{k/2} = 1^{k/2} = 1 \mod p^\alpha$$

  Como r es el orden del elemento g^k, entonces necesariamente $r|\varphi(p^\alpha)/2$. Ya que 2^{d-1} es la mayor potencia de 2 que divide a $\varphi(p^\alpha)/2$, de aquí deducimos que 2^d no puede dividir a r.

Como exactamente la mitad de elementos de $(\mathbb{Z}/p^\alpha\mathbb{Z})^*$ se expresan con k par y la mitad con k impar, esto concluye la demostración. □

Ahora podemos demostrar el Teorema 4.1.

Demostración. Probaremos que

$$P[r \text{ es impar ó } x^{r/2} = -1 \mod N] \leq \frac{1}{2^m}$$

Por el Teorema Chino de los Restos, elegir un elemento x aleatoriamente de $(\mathbb{Z}/N\mathbb{Z})^*$ es equivalente a elegir x_j aleatoriamente de $(\mathbb{Z}/p_j^{\alpha_j}\mathbb{Z})^*$, para $j = 1, \ldots, m$ tales que $x = x_j \mod p_j^{\alpha_j}$. Sea r_j el orden de x_j en $(\mathbb{Z}/p_j^{\alpha_j}\mathbb{Z})^*$, r el orden de x en $(\mathbb{Z}/N\mathbb{Z})^*$, 2^d la mayor potencia de 2 que divide a r y 2^{d_j} la mayor potencia de 2 que divide a r_j.

- Si es r impar, entonces ya que $r_j|r$ para cada j (porque $x_j^r = x^r \mod p_j^{\alpha_j}$. Como $x^r = 1 \mod N$, entonces $x^r = 1 \mod p_j^{\alpha_j}$. Entonces, $x_j^r = 1 \mod p_j^{\alpha_j}$ y de aquí el orden $r_j|r$); si r es impar entonces r_j es impar, y de aquí, $d_j = 0$ para todo j.

- Si r es par y es $x^{r/2} = -1 \mod N$, entonces $x^{r/2} = -1 \mod p_j^{\alpha_j}$, esto es, $x_j^{r/2} = -1 \mod p_j^{\alpha_j}$. De aquí, $r_j \not| (r/2)$. Ya que $r_j | r$, necesariamente $d = d_j$ para cada j (porque la mayor potencia de 2 que divide a r_j será exactamente 2^d, no puede ser menor porque $r_j \not| (r/2)$).

En definitiva, hemos probado que si r es impar ó $x^{r/2} = -1 \mod N$, entonces d_j toma el mismo valor para cada j. Esto implica que cada d_j debe dividir (o no dividir) a cada r_j simultáneamente. Por el Lema 4.1, la probabilidad de que esto ocurra es

$$\frac{1}{2} \cdot \frac{1}{2} \cdots \frac{1}{2} = \frac{1}{2^m}$$

Por contención de sucesos, finalmente

$$P[r \text{ es impar ó } x^{r/2} = -1 \mod N] \leq P[d_j = k \text{ para cada } j] \leq P[d_j | r_j \text{ ó } d_j \not| r_j \text{ para cada } j] = \frac{1}{2^m}$$

□

Una consecuencia inmediata del Teorema 4.1 es que si el número que queremos factorizar N tiene 2 factores primos (como suele usarse en criptografía, como es el caso del algoritmo RSA), entonces tenemos la garantía de que usando el procedimiento descrito anteriormente, daremos al menos un 75 % de las veces con un elemento y que nos permita hallar los factores de N usando la Proposición 4.1. Un resumen de un algoritmo para factorizar sería el siguiente:

1. Si N es par, devolver el factor 2.
2. Elegir aleatoriamente y entre 2 y $N - 2$. Si $\mathrm{mcd}(y, N) > 1$, devolver el factor $\mathrm{mcd}(y, N)$.
3. Encontrar el orden r del elemento y módulo N.
4. Si r es par y $x^{r/2} \neq -1 \mod N$, entonces calcular $\mathrm{mcd}(x^{r/2} - 1, N)$ y $\mathrm{mcd}(x^{r/2} + 1, N)$, comprobar cuál de ellos es un factor no trivial y devolver dicho factor. En caso contrario, volver al paso 2.

Gracias al algoritmo de Euclides y la exponenciación modular, todos los pasos de este algoritmo se pueden ejecutar en tiempo polinomial excepto el paso 3. Este problema se conoce como el problema de encontrar el orden, y no existe ningún método eficiente para resolverlo en un ordenador clásico. Sin embargo, usando la computación cuántica, sí es

posible resolver este problema eficientemente. En realidad, el algoritmo descrito es un esquema del algoritmo de Shor, con la salvedad de que el paso 3 se resuelve con un circuito cuántico, se obtienen los resultados y se prosigue en un ordenador clásico para culminar con la obtención del factor. Veremos cómo se construye este circuito en un ordenador cuántico y por qué funciona.

4.2 Computación cuántica

4.2.1 Bits y qubits

Ahora que hemos visto la motivación por la que necesitamos un ordenador cuántico, trataremos de describir cómo funciona. En el corazón de la computación clásica se encuentra el concepto de *bit*, la unidad mínima de información que describe un sistema clásico 2-dimensional. Matemáticamente, podemos modelar un bit como un elemento de $\mathbb{Z}_2 = \{0, 1\}$, y n bits como un elemento de $\mathbb{Z}_2^n$, donde la coordenada i-ésima representa el valor del bit i-ésimo. De este concepto surgen las *puertas lógicas*, mecanismos que convierten un conjunto de bits en otro. Matemáticamente, estas puertas lógicas se modelan como funciones $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^m$.

Ejemplo 4.1. La puerta lógica AND implementa la operación lógica de conjunción (&) y se define de esta manera:

$$f : \mathbb{Z}_2^2 \to \mathbb{Z}_2$$
$$(0,0) \longmapsto 0$$
$$(0,1) \longmapsto 0$$
$$(1,0) \longmapsto 0$$
$$(1,1) \longmapsto 1$$

La computación cuántica surge como una generalización de estos conceptos. De igual manera que en la computación clásica, en la computación cuántica se encuentra el concepto de *qubit*.

Definición 4.1. Un *qubit* es la unidad mínima de información describiendo un sistema cuántico 2-dimensional.

Un qubit se modela como un elemento del espacio de Hilbert $\mathbb{C}^2$. Como ya vimos, este es un espacio vectorial y posee una base ortonormal $\{e_1, e_2\}$ (por ejemplo, la canónica: $\{(0,1),(1,0)\}$). Así, todo qubit puede expresarse como una combinación $\mathbb{C}$-lineal de los elementos de la base. En el contexto de la computación cuántica, a estos elementos es usual denotarlos usando la notación de Dirac de esta manera:

$$|0\rangle = e_1$$

$$|1\rangle = e_2$$

y se denominan los *estados básicos*. Con lo que cualquier qubit puede expresarse de la forma $c_1 |0\rangle + c_2 |1\rangle$, $c_1, c_2 \in \mathbb{C}$. Una restricción importante que se impone sobre los qubits es que para todo qubit $x \in \mathbb{C}^2$, $\|x\|^2 = |c_1|^2 + |c_2|^2 = 1$ (lo cual ocurre si y sólo si $\|x\| = 1$, ver ejemplo 3.2). Así, cuando $c_1 = 1$ y $c_2 = 0$ se dice que el qubit está en el estado básico 0 y al contrario, en el estado básico 1. En cualquier otra situación, el qubit se dice que está en estado de *superposición*.

Los qubits pueden ser, en cualquier momento, "observados". Esto saca al qubit de su estado de superposición, y hace que se comporte como un bit clásico, tomando el valor 0 o el valor 1 de manera aleatoria. La probabilidad con la que toma cada valor viene dado por las coordenadas del qubit: así, un qubit $c_1 |0\rangle + c_2 |1\rangle$ tiene una probabilidad $|c_1|$ de valer 0 al ser observado, y $|c_2|$ de valer 1 (de aquí la imposición anterior de que $|c_1|^2 + |c_2|^2 = 1$).

Varios qubits se modelan matemáticamente usando el producto tensorial $(\mathbb{C}^2)^{\otimes n}$, donde n es el número de qubits. Así, 2 qubits se expresan de la forma $x = c_{00} |0\rangle \otimes |0\rangle + c_{01} |0\rangle \otimes |1\rangle + c_{01} |1\rangle \otimes |0\rangle + c_{11} |1\rangle \otimes |1\rangle$, $c_{00}, c_{01}, c_{10}, c_{11} \in \mathbb{C}$. En general, n qubits se expresan con 2^n coordenadas (pues la dimensión del producto tensorial de dos espacios vectoriales de dimensión m y n es mn). En la notación de Dirac es usual simplificar la notación del producto tensorial escribiendo simplemente $|x\rangle \otimes |y\rangle = |x\rangle |y\rangle = |xy\rangle$. En particular, cuando denotemos productos tensoriales de estados básicos, se escribirá $|i_1\rangle |i_2\rangle = |i_1 i_2\rangle$, $i_1, i_2 \in \{0, 1\}$.

La observación de varios qubits es análoga a la situación anterior. Por ejemplo, para 2 qubits, tendríamos que hay una probabilidad $|c_{00}|$ de medir un 0 en el primer qubit y un 0

en el segundo, una probabilidad $|c_{10}|$ de medir un 1 en el primero y un 0 en el segundo, y así sucesivamente. Sin embargo, hay que justificar que estas probabilidades efectivamente suman 1.

Proposición 4.2. *Sean* $|x_1\rangle, \ldots, |x_n\rangle$ *n qubits, con* $|x_i\rangle = x_0^{(i)}|0\rangle + x_1^{(i)}|1\rangle$. *Entonces,* $|x_1 x_2 \cdots x_n\rangle$ *verifica que*

$$\| |x_1 x_2 \cdots x_n\rangle \|^2 = \left\| \sum_{i_1,\ldots,i_n \in \{0,1\}} \left(\left(\prod_{j=1}^{n} x_{i_j}^{(j)} \right) |i_1 i_2 \cdots i_n\rangle \right) \right\|^2 = 1$$

Demostración. Lo demostraremos por inducción. Para $n = 1$ el resultado se tiene por definición de qubit.

Supongamos el resultado cierto para $n = k$, y denotemos por simplicidad

$$|x_1 \ldots x_k\rangle = \alpha_0 |00\cdots0\rangle + \alpha_1 |00\cdots1\rangle + \cdots + \alpha_{2^k-1} |11\cdots1\rangle$$

Entonces, si $|x_{k+1}\rangle = \beta_0 |0\rangle + \beta_1 |1\rangle$,

$$\begin{aligned} |x_1 \ldots x_k\rangle |x_{k+1}\rangle = \beta_0(\alpha_0 |00\cdots00\rangle + \alpha_1 |00\cdots10\rangle + \cdots + \alpha_{2^k-1} |11\cdots10\rangle) + \\ + \beta_1(\alpha_0 |00\cdots01\rangle + \alpha_1 |00\cdots11\rangle + \cdots + \alpha_{2^k-1} |11\cdots11\rangle) \end{aligned}$$

Por tanto sacando factor común,

$$\| |x_1 \ldots x_{k+1}\rangle \|^2 = |\beta_0|^2 \left(\sum_{j=0}^{2^k-1} |\alpha_j|^2 \right) + |\beta_1|^2 \left(\sum_{j=0}^{2^k-1} |\alpha_j|^2 \right)$$

Por hipótesis de inducción, $(\sum_{j=0}^{2^k-1} |\alpha_j|^2) = 1$ y además $|\beta_0|^2 + |\beta_1|^2 = 1$ por ser $|x_{k+1}\rangle$ qubit, así,

$$\| |x_1 \ldots x_{k+1}\rangle \|^2 = |\beta_0|^2 + |\beta_1|^2 = 1$$

□

4.2.2 Puertas cuánticas y circuitos cuánticos

Una vez que hemos definido el concepto de qubit y cómo se modelan matemáticamente, el siguiente paso es definir el concepto de puerta cuántica; que no es más que una función

$f : \mathbb{C}^{\otimes 2n} \to \mathbb{C}^{\otimes 2n}$ (envía n qubits a n qubits). Ya que, como vimos, estos dos espacios son espacios vectoriales, podemos tratar f como una aplicación lineal. Además, ya que f debe enviar qubits a qubits, debe verificarse para todo $x \in \mathbb{C}^{\otimes 2n}$, que

$$\|f(x)\| = \|x\| = 1$$

Y según vimos en el capítulo anterior, esto se logra si y sólo si f es unitaria. Esta condición implica que el operador dado por $A^\dagger$ es la inversa de A. Es decir, toda puerta cuántica debe ser reversible (biyectiva), y su inversa está dada por la matriz $A^\dagger$. Es por ello que toda puerta cuántica debe tener exactamente los mismos qubits de entrada y de salida.

Ejemplo 4.2. Las puertas cuánticas generalizan las puertas clásicas. Por ejemplo, la puerta $NOT : \mathbb{C}^2 \otimes \mathbb{C}^2 \to \mathbb{C}^2 \otimes \mathbb{C}^2$ es aquella puerta cuya matriz asociada es

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Este operador es unitario, puesto que

$$AA^\dagger = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = Id$$

Además, verifica que

$$NOT\,|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle$$

$$NOT\,|1\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle$$

Ejemplo 4.3. Un ejemplo importante en la computación cuántica es el operador de Hadamard, aquel dado por la matriz

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

No es difícil ver que este operador es unitario y que verifica las siguientes identidades:

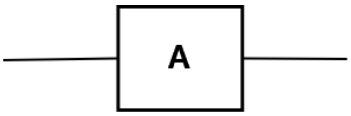
$$H\,|0\rangle = \frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle$$

$$H\,|1\rangle = \frac{1}{\sqrt{2}}\,|0\rangle - \frac{1}{\sqrt{2}}\,|1\rangle$$

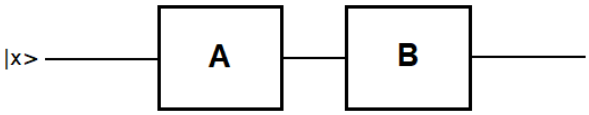
$$H\left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right) = |0\rangle$$
$$H\left(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle\right) = |1\rangle$$

Este operador será sumamente útil, porque nos permite convertir qubits en estados básicos a qubits en estados de superposición equiprobables (50 % de medir 0, 50 % de medir 1) y viceversa.

Una sucesión de puertas cuánticas actuando sobre un conjunto de qubits forma un circuito. Para describir fácilmente circuitos que implementen algoritmos cuánticos, es común representarlos usando esquemas. Una puerta cuántica A actuando sobre un qubit se representa de esta manera:

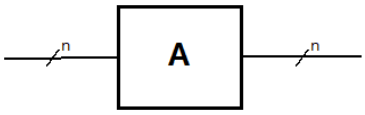


Normalmente, por convenio se supone que la computación ocurre de izquierda a derecha. De esta manera, el siguiente circuito indica la operación de aplicar A y luego B sobre un qubit $|x\rangle$, es decir, $B(A(x))$:



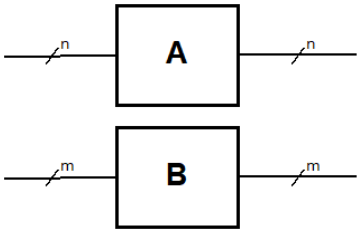
Estas dos operaciones se dice que son *secuenciales*, porque ocurre una después de la otra. Notemos que podríamos haber simplificado este circuito calculando el operador $C = B \circ A$, cuya matriz asociada viene dada de multiplicar BA, las matrices asociadas de B y de A, y reescribiendo el circuito con un solo operador actuando sobre $|x\rangle$, C.

Cuando tengamos un operador tenga más de un qubit de entrada o salida, en lugar de escribir muchas líneas, lo denotamos de esta manera:



En este caso, A sería un operador que trabaja sobre n qubits (observemos que hubiese sido equivalente haber dibujado n líneas entrando y saliendo de A).

En un circuito cuántico, también pueden ocurrir operaciones *en paralelo* sobre varios qubits. Por ejemplo:



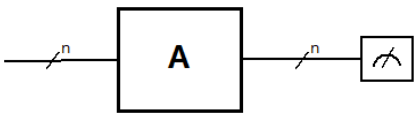
Si $|x\rangle$,$|y\rangle$ son n y m qubits respectivamente, Entonces este circuito aplica, sobre la entrada $|x\rangle \otimes |y\rangle$, la operación $A|x\rangle \otimes B|y\rangle$. Este circuito podría simplificarse calculando el operador $C = A \otimes B$ (ver apéndice), cuya entrada son $n + m$ qubits, de manera que la acción del circuito podría escribirse como $C(|x\rangle \otimes |y\rangle) = (A \otimes B)(|x\rangle \otimes |y\rangle)$.

Ejemplo 4.4. El operador de Hadamard H aplicado sobre varios qubits se comporta como sigue. Dados n qubits $|x_1 \cdots x_n\rangle$ en estado básico 0, entonces el resultado de aplicar el operador H a cada uno de ellos en paralelo es:

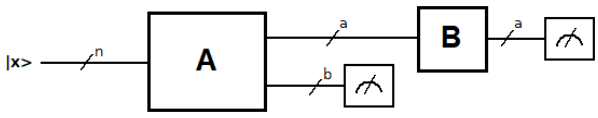
$H|x_1\rangle \otimes H|x_2\rangle \otimes \cdots \otimes H|x_n\rangle = H|0\rangle \otimes H|0\rangle \otimes \cdots \otimes H|x_n\rangle = \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right) \otimes \cdots \otimes \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right) = \frac{1}{\sqrt{2^n}} \sum_{i_1,\ldots,i_n \in \{0,1\}} |i_1 \cdots i_n\rangle$.

Por simplicidad, este operador se denota $H^{\otimes n}$, y su matriz asociada es $\frac{1}{\sqrt{2^n}} H_{2^n}$, donde H_{2^n} es la matriz de Hadamard de orden 2^n **(añadir referencia)**.

Una puerta especial es la de *medición*. Cuando se coloca al final de uno o varios qubits, pasan de ser un bits cuánticos a bits clásicos siguiendo la ley de probabilidad explicada anteriormente. En el siguiente ejemplo, se medirían simultáneamente n qubits tras aplicarles un operador A:



Un caso interesante es cuando se miden qubits parcialmente, por ejemplo, usando este circuito:



Donde, obviamente, $a + b = n$. En este caso, el estado de $|x\rangle$ justo después de aplicarse A es $A(|x\rangle)$. Sin embargo, después se observan los últimos b qubits de $A(|x\rangle)$; ¿cuál será entonces la entrada de B, que se aplica sobre los restantes a qubits, que no fueron observados?

Denotemos la salida de A como

$$|x_1\rangle = |y_1\rangle\,|z_1\rangle = \sum_{i=0}^{2^a-1}\sum_{j=0}^{2^b-1}\alpha_{ij}\,|ij\rangle$$

Si tras medir $|z_1\rangle$ observamos el estado $|k\rangle$, $k \in [0, 2^b - 1]$ (el cual ocurrirá con probabilidad $\sum_{i=0}^{2^a-1}|\alpha_{ik}|^2$), entonces parece lógico que el estado final del sistema tras medir los últimos b qubits sea

$$|y\rangle = \sum_{i=0}^{2^a-1}\alpha_{ik}\,|ik\rangle$$

Sin embargo, el módulo de este qubit no es necesariamente 1 (en general, $|y\rangle$ no es el producto tensorial de varios qubits). Por eso, es necesario *renormalizar* el qubit:

$$|y\rangle = \frac{\sum_{i=0}^{2^a-1}\alpha_{ik}\,|ik\rangle}{\sqrt{\sum_{i=0}^{2^a-1}|\alpha_{ik}|^2}}$$

Puesto que ahora los últimos b qubits son siempre constantes y valen k, podemos ignorarlos, ya que ahora se comportan como bits clásicos, y finalmente nos queda que la entrada de B es:

$$|y\rangle = \frac{\sum_{i=0}^{2^a-1}\alpha_{ik}\,|i\rangle}{\sqrt{\sum_{i=0}^{2^a-1}|\alpha_{ik}|^2}}$$

La justificación formal de estas ideas intuitivas se apoya en la teoría de operadores de medida (ver [8]).

Ejemplo 4.5. Con la notación anterior, si $\alpha_{ik} = \frac{1}{\sqrt{2^n}}$ para n amplitudes y valen 0 en las demás, entonces

$$|y\rangle = \frac{\frac{1}{\sqrt{2^n}}\sum_i |i\rangle}{\sqrt{\sum_i \left|\frac{1}{\sqrt{2^n}}\right|^2}} = \frac{\sum_i |i\rangle}{\sqrt{n}}$$

Es decir, el resultado de medir un subconjunto de un conjunto de qubits en estado de superposición equiprobable, es otro estado de superposición equiprobable, cuya expresión

es la suma de todos los posibles estados de los qubits no medidos y con amplitudes el inverso de la raíz cuadrada del número de estados posibles.

4.2.3 El circuito para encontrar el orden

Con estas nociones básicas, finalmente podemos presentar el circuito para resolver el problema de encontrar el orden. Con tal de dar una visión general del algoritmo, supondremos que r es una potencia de 2. El caso general requiere usar un algoritmo de fracciones continuas (ver [8]) para poder hallar r en el caso general. El circuito, no obstante, no varía; solo se trata de un paso extra al final del proceso.

Necesitaremos dos puertas cuánticas principalmente, además de la puerta de Hadamard $H^{\otimes n}$ presentada anteriormente. Una de ellas es la *transformada cuántica de Fourier*. Esta transformación actúa sobre los estados básicos $|0\rangle, \cdots, |N-1\rangle$ (por ejemplo, en 2 qubits, $N = 4$) de la siguiente forma: para cada $0 \leq j \leq N-1$

$$|j\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi ijk/N} |k\rangle$$

La definición de la transformada cuántica de Fourier (abreviadamente, *QFT*, de *Quantum Fourier Transform*) sobre un qubit se extiende por linealidad. Para que sea una puerta cuántica, debe ser un operador unitario.

Proposición 4.3. *La transformada cuántica de Fourier es un operador unitario.*

Demostración. Denotemos $A = (a_{jk})$, $k, j \in \{0, \cdots, N-1\}$ la matriz asociada de QFT respecto de la base común de los espacios de llegada y salida $\{|0\rangle, \ldots, |N-1\rangle\}$. Entonces, denotando π^j la proyección sobre $|j\rangle$:

$$a_{jk} = \pi^j(QFT(|k\rangle)) = \frac{1}{\sqrt{N}} e^{2\pi ijk/N}$$

Por otro lado, denotando $A^\dagger = (b_{jk})$, $k, j \in \{0, \cdots, N-1\}$, entonces:

$$b_{jk} = \overline{a_{kj}} = \frac{1}{\sqrt{N}} \overline{e^{2\pi ijk/N}} = \frac{1}{\sqrt{N}} e^{-2\pi ijk/N}$$

Veamos que $A^\dagger A = I$. Denotemos c_{kj} las entradas de la matriz $A^\dagger A$.

- Si $k = j$, entonces:

$$c_{kk} = \sum_{j=0}^{N-1} b_{kj} a_{jk} = \sum_{j=0}^{N-1} \left(\frac{1}{\sqrt{N}} e^{-2\pi i k j/N} \right) \left(\frac{1}{\sqrt{N}} e^{2\pi i j k/N} \right) = \sum_{j=0}^{N-1} \frac{1}{N} = N \frac{1}{N} = 1$$

- Si $k \neq j$, entonces $k - j \in \mathbb{Z} \setminus \{0\}$. Por tanto,

$$c_{kl} = \sum_{j=0}^{N-1} b_{lj} a_{jk} = \left(\frac{1}{\sqrt{N}} e^{-2\pi i l j/N} \right) \left(\frac{1}{\sqrt{N}} e^{2\pi i j k/N} \right) =$$

$$= \frac{1}{N} \sum_{j=0}^{N-1} e^{\frac{2\pi i j}{N}(k-l)} = \frac{1}{N} \frac{1 - \left(e^{\frac{2\pi i j}{N}(k-l)} \right)^N}{1 - e^{\frac{2\pi i}{N}(k-l)}}$$

  Puesto que $k - l \in \mathbb{Z}$, $\left(e^{\frac{2\pi i j}{N}(k-l)} \right)^N = e^{2\pi i j (k-l)} = 1$ y esto implica que $c_{kl} = 0$.

Así pues, como $c_{kj} = \begin{cases} 1 \text{ si } k = j \\ 0 \text{ si } k \neq j \end{cases}$ $A^\dagger A = I$, como queríamos demostrar.

□

La otra puerta cuántica clave en la construcción del circuito es la siguiente. Dada $f : \mathbb{Z}_2^m \to \mathbb{Z}_2^n$ una función clásica, se define el operador U_f definido sobre $m+n$ qubits como:

$$|x, y\rangle \mapsto |x, y \oplus f(x)\rangle$$

donde $\oplus$ denota la suma en $\mathbb{Z}^2$ bit a bit. Es posible ver que este operador es unitario, y por tanto, es una puerta cuántica válida, para cualquier f ([8]). La gran utilidad de esta puerta viene cuando el primer conjunto de qubits, x, está en un estado de superposición equiprobable y el segundo, y, vale $|0\rangle$:

$$U_f \left(\frac{1}{\sqrt{2^m}} \sum_{j=0}^{2^m-1} |j, 0\rangle \right) = \frac{1}{\sqrt{2^m}} \sum_{j=0}^{2^m-1} |j, 0 \oplus f(j)\rangle = \frac{1}{\sqrt{2^m}} \sum_{j=0}^{2^m-1} |j, f(j)\rangle$$

Como podemos ver, la puerta cuántica U_f nos permite hacer 2^m evaluaciones simultáneas de f, con cada valor en un estado de superposición equiprobable junto con los valores de entrada. Es posible ver que U_f puede ser ejecutada en un tiempo eficiente. Esto no se puede conseguir en general en un ordenador clásico, y es por ello que este algoritmo sólo puede

ser ejecutado en un ordenador cuántico.

La función que nos interesa en nuestro caso es, dado N el número a factorizar y a un número coprimo con N, la función definida para cada $x \in \mathbb{Z}$ por

$$f_{a,N}(x) = a^x \mod N$$

Queremos, pues, encontrar el periodo de esta función, es decir, encontrar el menor $r \in \mathbb{N}$ tal que para todo $x \in \mathbb{Z}$,

$$f_{a,N}(r+x) = f_{a,N}(x)$$

Esta función devuelve siempre un número menor que N, por tanto, necesita $n = \log_2 N$ bits de salida. Para encontrar el periodo con seguridad, es necesario evaluar $f_{a,N}$ para todo $0 \leq x \leq N^2$ (en el peor caso, el periodo es $r = N - 1$). Por tanto, necesitaremos $m = \log_2 N^2 = 2\log_2 N = 2n$ qubits de entrada.

El circuito que nos permite hallar el orden es el siguiente.

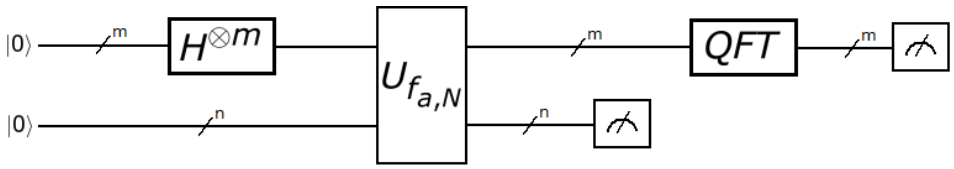


Figura 4.1: El circuito para encontrar el periodo de la función $f_{a,N}$.

Veamos cómo va evolucionando el sistema tras cada aplicación de cada puerta cuántica.

1. Comenzamos con los m y n qubits en estado $|0\rangle$, es decir, el estado del sistema es $|0,0\rangle$.

2. Aplicamos una puerta Hadamard a los primeros m qubits, obteniendo el estado

$$\left(H^{\otimes m}|0\rangle\right) \otimes |0\rangle = \frac{1}{\sqrt{2^m}} \sum_{j=0}^{2^m-1} |j,0\rangle$$

3. Aplicamos ahora el operador $U_{f_{a,N}}$, obteniendo:

$$U_{f_{a,N}} \left(\frac{1}{\sqrt{2^m}} \sum_{j=0}^{2^m-1} |j,0\rangle \right) = \frac{1}{\sqrt{2^m}} \sum_{j=0}^{2^m-1} |j, f_{a,N}(j)\rangle = \frac{1}{\sqrt{2^m}} \sum_{j=0}^{2^m-1} \left|j, a^j \mod N\right\rangle$$

4. Medimos los últimos n qubits del sistema. Por tanto, mediremos $\alpha = a^{\overline{x}}$ mód N, para cierto $0 \leq \overline{x} \leq N-1$. Sin embargo, como r es el orden del elemento a, entonces se verifica que $a^{\overline{x}+kr} = a^{\overline{x}}$ mód N para todo $k \in \mathbb{Z}$. En total, de los 2^m valores en los que hemos evaluado f, en total habrá

$$\left\lfloor \frac{2^m}{r} \right\rfloor$$

valores de x que son iguales a α, módulo N.

Tal y como habíamos anunciado, haremos la asunción que r divide a 2^m con el objetivo de simplificar los resultados de aquí en adelante. Por tanto, el número de valores es exactamente $\frac{2^m}{r}$. Siguiendo pues el razonamiento del Ejemplo 4.5, el estado del sistema será

$$\frac{\sum_{\{x:a^x=\alpha \text{ mód } N\}} |x,\alpha\rangle}{\sqrt{\frac{2^m}{r}}}$$

Por la periodicidad de $f_{a,N}$, podemos reescribir el estado anterior observando que podemos expresar los $\frac{2^m}{r}$ valores como t_0+kr para cierto $t_0 \in \mathbb{Z}$ y $k = 0, \cdots, \frac{2^m}{r}-1$:

$$\frac{\sum_{k=0}^{\frac{2^m}{r}-1} |t_0+kr,\alpha\rangle}{\sqrt{\frac{2^m}{r}}}$$

Los últimos n qubits ya pueden ser descartados puesto que ya han sido medidos, quedándonos los m qubits en el siguiente estado en superposición:

$$\frac{\sum_{k=0}^{\frac{2^m}{r}-1} |t_0+kr\rangle}{\sqrt{\frac{2^m}{r}}}$$

5. Aplicamos QFT. Denotaremos $A = \frac{1}{\sqrt{\frac{2^m}{r}}}$.

$$QFT\left(\frac{\sum_{k=0}^{\frac{2^m}{r}-1}|t_0+kr\rangle}{\sqrt{\frac{2^m}{r}}}\right)=A\sum_{k=0}^{\frac{2^m}{r}-1}\sum_{j=0}^{2^m-1}e^{\frac{2\pi i(t_0+kr)j}{2^m}}|j\rangle=$$
$$=A\sum_{j=0}^{2^m-1}\sum_{k=0}^{\frac{2^m}{r}-1}e^{\frac{2\pi i(t_0+kr)j}{2^m}}|j\rangle=A\sum_{j=0}^{2^m-1}e^{\frac{2\pi it_0j}{2^m}}\left(\sum_{k=0}^{\frac{2^m}{r}-1}e^{\frac{2\pi ikrj}{2^m}}\right)|j\rangle \tag{4.2}$$

Veamos cuánto vale el sumatorio entre paréntesis de la expresión (4.2).

- Si j es un múltiplo de $\frac{2^m}{r}$, es decir, $j=\lambda\frac{2^m}{r}$, entonces

$$\sum_{k=0}^{\frac{2^m}{r}-1}e^{\frac{2\pi ikrj}{2^m}}=\sum_{k=0}^{\frac{2^m}{r}-1}e^{\frac{2\pi ikr\lambda\frac{2^m}{r}}{2^m}}=\sum_{k=0}^{\frac{2^m}{r}-1}e^{2\pi ik\lambda}=\sum_{k=0}^{\frac{2^m}{r}-1}1=\frac{2^m}{r}$$

- En caso contrario, entonces como $e^{2\pi i\frac{r}{2^m}j}\neq 1$ al no cancelarse $\frac{r}{2^m}\notin\mathbb{Z}$:

$$\sum_{k=0}^{\frac{2^m}{r}-1}e^{\frac{2\pi ikrj}{2^m}}=\frac{1-\left(e^{\frac{2\pi irj}{2^m}}\right)^{\frac{2^m}{r}}}{1-e^{\frac{2\pi irj}{2^m}}}=\frac{1-e^{2\pi ij}}{1-e^{\frac{2\pi irj}{2^m}}}=0$$

Por tanto, podemos reescribir (4.2) como

$$A\sum_{\lambda=0}^{\lfloor r/2^m-1\rfloor}\frac{2^m}{r}e^{\frac{2\pi it_0\lambda\frac{2^m}{r}}{2^m}}\left|\lambda\frac{2^m}{r}\right\rangle=A\sum_{\lambda=0}^{\lfloor r/2^m-1\rfloor}\frac{2^m}{r}e^{\frac{2\pi it_0\lambda}{r}}\left|\lambda\frac{2^m}{r}\right\rangle \tag{4.3}$$

6. Medimos los m qubits. A partir de la ecuación (4.3) es fácil ver que al medir, obtendremos

$$x=k\frac{2^m}{r}$$

Para cierto $k\in\mathbb{Z}$. Entonces,

$$\frac{x}{2^m}=\frac{k}{r}$$

El miembro de la derecha de la igualdad es conocido, por tanto, basta reducir esta fracción a su forma irreducible para obtener el valor de r, terminando así el algoritmo.

4.3 El algoritmo de Grover

Finalizamos este capítulo con el otro algoritmo cuántico que acelera otro gran problema: el problema de la búsqueda. Como ya adelantamos al principio, el problema de la búsqueda puede formularse de la siguiente manera: dado un conjunto S de cardinal N y una función $f : S \to \{0, 1\}$, encontrar $x \in S$ tal que $f(x) = 1$. En un ordenador clásico, este problema puede resolverse en $\mathcal{O}(N)$ operaciones, evaluando cada elemento de S mediante f. Gracias al siguiente algoritmo cuántico que veremos, podremos resolver este problema en $\mathcal{O}(\sqrt{N})$ operaciones, una reducción cuadrática de complejidad.

4.3.1 Introducción

Primero, comenzaremos con una traducción del problema a términos de computación cuántica. En vez de buscar entre los elementos de S directamente, ya que $|S| = N < +\infty$, podemos asignar a cada elemento de S un índice numerado entre 0 y $N - 1$. Por tanto, buscaremos entre dichos índices la solución deseada. Denotaremos también por $0 < M \leq N$ el número de elementos de S que son solución del problema de la búsqueda, es decir, aquellos elementos que verifican que $f(x) = 1$.

La puerta cuántica clave en el que se basa este algoritmo es el *oráculo* (O). Esta puerta, en realidad, es la misma que la puerta U_f del algoritmo de Shor: actúa sobre los estados básicos de la forma

$$|x\rangle\,|q\rangle \xrightarrow{O} |x\rangle\,|q \oplus f(x)\rangle$$

para cada $x \in \{0, \ldots, N-1\}$, $q \in \{0, 1\}$. Por tanto, para comprobar si cierto x es solución del problema de búsqueda, sólo hace falta aplicar el oráculo a $|x\rangle\,|0\rangle$, medir el último qubit y ver si vale 0 ó 1.

Proposición 4.4. *Si $|q\rangle = H(|1\rangle)$, entonces la acción del oráculo sobre $|x\rangle\,|q\rangle$ para cada $x \in \{0, \ldots, N-1\}$ es $(-1)^{f(x)}\,|x\rangle\,H(|1\rangle)$, donde H denota la puerta de Hadamard.*

Demostración. Recordemos que $H(|1\rangle) = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ (ver Ejemplo 4.3). Por tanto,

$$O\left(|x\rangle\left(\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right)\right) = O\left(\frac{1}{\sqrt{2}}|x\rangle|0\rangle\right) - O\left(\frac{1}{\sqrt{2}}|x\rangle|1\rangle\right)$$

Ahora distinguimos dos casos:

1. Si x es una solución del problema de búsqueda, $O(|x\rangle|0\rangle) = |x\rangle|1\rangle$ y $O(|x\rangle|1\rangle) = |x\rangle|0\rangle$. Por tanto,

$$O\left(|x\rangle\left(\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right)\right) = \frac{1}{\sqrt{2}}|x\rangle|1\rangle - \frac{1}{\sqrt{2}}|x\rangle|0\rangle = -|x\rangle\left(\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right)$$

2. Si x no es una solución del problema de búsqueda, $O(|x\rangle|0\rangle) = |x\rangle|0\rangle$ y $O(|x\rangle|1\rangle) = |x\rangle|1\rangle$. En este caso,

$$O\left(|x\rangle\left(\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right)\right) = \frac{1}{\sqrt{2}}|x\rangle|0\rangle - \frac{1}{\sqrt{2}}|x\rangle|1\rangle = |x\rangle\left(\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right)$$

En conclusión, si $f(x) = 1$, entonces $O(|x\rangle H(|1\rangle)) = (-1)^1|x\rangle H(|1\rangle)$, y si $f(x) = 0$, entonces $O(|x\rangle H(|1\rangle)) = (-1)^0|x\rangle H(|1\rangle)$, lo que prueba el resultado. □

Ya que el segundo qubit se mantiene constante, podemos simplificar la notación diciendo que el oráculo actúa sobre $|x\rangle$ devolviendo $(-1)^{f(x)}|x\rangle$.

4.3.2 El circuito cuántico del algoritmo

En lo que sigue, supondremos que $N = 2^n$ para cierto $n \in \mathbb{N}$, con lo que cada elemento de S puede expresarse con n qubits ($|0\rangle, \ldots, |N-1\rangle$). El esquema del circuito cuántico para el algoritmo de Grover se resume en la siguiente figura.

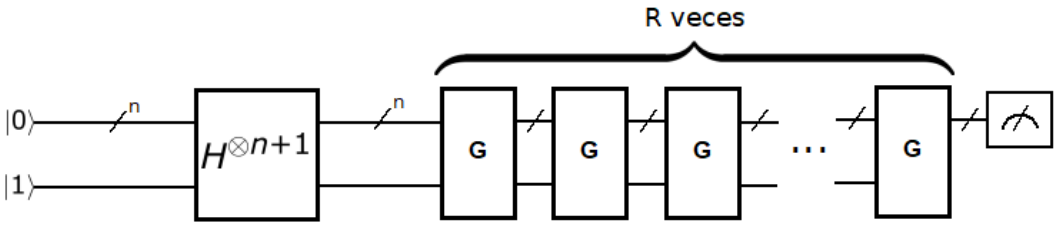


Figura 4.2: El esquema general del circuito para ejecutar el algoritmo de Grover.

En este caso, G es una puerta cuántica, llamada *operador de Grover* o *iteración de Grover*, que se ejecuta R veces (más adelante, daremos cotas para este valor en función de la probabilidad de éxito del algoritmo deseada). Su estructura es la siguiente:

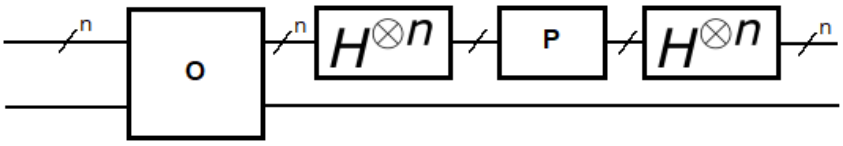


Figura 4.3: El esquema del operador de Grover.

Donde O es el oráculo del problema, y P es una puerta cuántica que actúa sobre los estados básicos $|0\rangle, \ldots, |N-1\rangle$ de la siguiente manera:

$$|0\rangle \xrightarrow{P} |0\rangle$$

$$|x\rangle \xrightarrow{P} -|x\rangle \text{ para todo } x > 0$$

Proposición 4.5. *El operador P actuando sobre n qubits viene dado por la matriz*

$$Q = 2M - I$$

donde $M = (m_{ij})_{i,j\in\{1,\ldots,2^n\}}$ es una matriz dada por $m_{11} = 1$ y 0 para el resto de entradas.

Demostración. Veremos primero que el operador dado por la matriz Q es unitario: en efecto, ya que $M^\dagger = M$,

$$Q^\dagger Q = (2M^\dagger - I)(2M - I) = 4M^2 - 4M + I$$

Por otro lado, es fácil ver que $M^2 = M$. Por tanto, $4M^2 - 4M + I = 4M - 4M + I = I$, y Q es un operador unitario.

Comprobemos, finalmente, que cumple las propiedades buscadas:

$$Q(|0\rangle) = \begin{pmatrix} 1 & 0 & \ldots & 0 \\ 0 & -1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = |0\rangle$$

$$Q(|x\rangle) = \begin{pmatrix} 1 & 0 & \ldots & 0 \\ 0 & -1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & -1 \end{pmatrix} \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ -1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = -|x\rangle$$

Donde $x > 0$, lo cual concluye la demostración. □

4.3.3 Punto de vista geométrico y tiempos de ejecución

Para poder dar cotas para R, es conveniente ver el algoritmo desde un punto de vista geométrico [8]. Los n qubits que se pasan a la primera iteración de Grover G tienen un estado inicial equiprobable $|\psi\rangle$. Denotaremos por $\sum_x'$ una suma sobre los elementos que son solución del problema de búsqueda, y $\sum_x''$ los que no. Entonces, los siguientes estados

$$|\alpha\rangle = \frac{1}{\sqrt{N-M}} \sum_x'' |x\rangle$$

$$|\beta\rangle = \frac{1}{\sqrt{M}} \sum_x' |x\rangle$$

representan el conjunto de todas las no soluciones y soluciones del problema en estado equiprobable, respectivamente. Así, es fácil ver dividiendo la suma de todos los x de S en los dos sumandos y multiplicando y dividiendo por las cantidades correspondientes en cada caso, que el estado inicial $|\psi\rangle$ puede expresarse como

$$|\psi\rangle = \sqrt{\frac{N-M}{N}} |\alpha\rangle + \sqrt{\frac{M}{N}} |\beta\rangle = a|\alpha\rangle + b|\beta\rangle \tag{4.4}$$

A partir de la Proposición 4.4 es fácil ver que la acción del oráculo sobre $|\psi\rangle$ es

$$O(|\psi\rangle) = O(a|\alpha\rangle + b|\beta\rangle) = a|\alpha\rangle - b|\beta\rangle$$

Esto corresponde a una reflexión del vector $|\psi\rangle$ en el plano generado por los vectores $|\alpha\rangle$ y $|\beta\rangle$ respecto del vector $|\alpha\rangle$. Es posible ver, con un poco más de trabajo, que el resto de la acción del operador de Grover hace una reflexión respecto del vector $|\psi\rangle$ en el plano generado por los vectores $|\alpha\rangle$ y $|\beta\rangle$ [8]. La composición de dos reflexiones es una rotación. Tomando θ de manera que

$$\cos\frac{\theta}{2} = \sqrt{\frac{N-M}{N}}$$

entonces

$$\sin\frac{\theta}{2} = \sqrt{1-\cos^2(\theta/2)} = \sqrt{1-\frac{N-M}{N}} = \sqrt{\frac{M}{N}}$$

Luego podemos expresar

$$|\psi\rangle = \cos\frac{\theta}{2}|\alpha\rangle + \sin\frac{\theta}{2}|\beta\rangle$$

Y mediante un sencillo argumento geométrico [8], se puede ver aplicando las reflexiones anteriormente citadas que

$$G(|\psi\rangle) = \cos\frac{3\theta}{2}|\alpha\rangle + \sin\frac{3\theta}{2}|\beta\rangle$$

Así que el operador G efectúa una reflexión de ángulo θ sobre la base $|\alpha\rangle$,$|\beta\rangle$. Si suponemos que $0 < \theta < \frac{\pi}{2}$ (es suficiente que $0 < M < N/2$), ya que el seno es creciente y el coseno es decreciente en $[0, \pi/2]$, las aplicaciones sucesivas de G llevan al estado del ordenador cada vez más próximo a $|\beta\rangle$. De esta manera, una observación de los qubits producirá uno de los resultados en superposición de $|\beta\rangle$ con mucha probabilidad, las cuales son todas soluciones del problema que buscábamos.

Ahora sólo queda calcular cuántas veces debemos rotar $|\psi\rangle$ mediante el ángulo θ para estar lo más cerca posible del vector $|\beta\rangle$, es decir, cuántas veces debemos aplicar G. A partir de la ecuación (4.4), vemos que una rotación de ángulo

$$\gamma = \frac{\pi}{2} - \arcsin\sqrt{\frac{M}{N}}$$

Llevaría $|\psi\rangle$ al estado $|\beta\rangle$ (el ángulo en coordenadas polares de $|\psi\rangle$ es $\operatorname{arc\,sen}\sqrt{\frac{M}{N}}$, sumar el complementario llevaría el estado a $0|\alpha\rangle + 1|\beta\rangle$). Desarrollando la expresión anterior, vemos que

$$\operatorname{arc\,sen}\sqrt{\frac{M}{N}} = \frac{\pi}{2} - \gamma$$

$$\sqrt{\frac{M}{N}} = \sin\left(\frac{\pi}{2} - \gamma\right) = \cos\gamma$$

$$\gamma = \operatorname{arc\,cos}\sqrt{\frac{M}{N}}$$

De aquí, denotando $CI(x)$ la función entero más próximo a $x \in \mathbb{R}$, el número de veces que debemos rotar $|\psi\rangle$ por el ángulo θ para estar más próximo al vector $|\beta\rangle$ es:

$$R = CI\left(\frac{\operatorname{arc\,cos}\sqrt{\frac{M}{N}}}{\theta}\right) \tag{4.5}$$

La expresión (4.5) nos da una expresión exacta para el número de iteraciones de Grover R que hay que realizar para maximizar la probabilidad de éxito, pero se puede hallar una cota superior que no depende de relaciones trigonométricas y de θ. Para ello, observemos

primero que la función $CI(x) \leq \lceil x \rceil$ para todo $x \in \mathbb{R}$, y que $0 \leq \arccos(\theta) \leq \frac{\pi}{2}$ para todo $0 \leq \theta \leq 1$. Por tanto como, $0 \leq \sqrt{\frac{M}{N}} \leq 1$:

$$R \leq \left\lceil \frac{\arccos \sqrt{\frac{M}{N}}}{\theta} \right\rceil \leq \left\lceil \frac{\pi}{2\theta} \right\rceil$$

Si suponemos que $M \leq \frac{N}{2}$, entonces al ser $0 < \theta < \frac{\pi}{2}$:

$$\frac{\theta}{2} \geq \sin \frac{\theta}{2} = \sqrt{\frac{M}{N}}$$

$$\frac{1}{\theta} \leq \frac{1}{2\sqrt{\frac{M}{N}}} = \frac{1}{2}\sqrt{\frac{N}{M}}$$

Luego

$$R \leq \left\lceil \frac{\pi}{4}\sqrt{\frac{N}{M}} \right\rceil$$

Es decir, se deben efectuar $\mathcal{O}\left(\sqrt{\frac{N}{M}}\right)$ iteraciones de Grover. En el caso particular de que $M = 1$, es decir, que haya una única solución al problema, entonces se deben efectuar $\mathcal{O}(\sqrt{N})$ iteraciones de Grover, como anunciamos al principio de esta sección.

Como observación final, en toda la discusión anterior hemos supuesto que $M \leq N/2$. En realidad, es razonable hacer esta suposición: en un problema de búsqueda suelen haber muy pocas soluciones en comparación con el cardinal del conjunto de búsqueda. Si así no fuera, simplemente podemos escoger un elemento al azar de S. Como más de la mitad de los elementos de S son soluciones del problema, la probabilidad de encontrar una solución realizando este procedimiento es, al menos, $\frac{1}{2}$. La repetición sucesiva de este procedimiento producirá una solución del problema muy rápidamente.

CAPITULO

5

Algoritmos de criptografía poscuántica

En esta sección presentamos varias propuestas, tanto de criptosistemas como de sistemas de firma, que se cree que resisten a los ordenadores cuánticos. La idea para desarrollar sistemas de clave pública es considerar problemas difíciles, es decir, problemas en los que no existan algoritmos clásicos ni cuánticos eficientes que los resuelvan. Hasta la aparición de los ordenadores cuánticos, los dos problemas más utilizados eran el problema de la factorización de números enteros (base del criptosistema y sistema de firma *RSA*) y el problema del logaritmo discreto (base de los sistemas de firma *DSA* y *ECDSA*). Sin embargo, al haberse encontrado algoritmos eficientes que los resuelven en ordenadores cuánticos, estos sistemas quedarán obsoletos cuando la computación cuántica sea una realidad.

Nuestro estudio se centrará en aquellos sistemas basados en funciones hash y aquellos basados en códigos autocorrectores. Estos se basan en dos problemas fundamentales: encontrar la preimagen de una función de un solo sentido (criptografía basada en funciones hash) y el problema de encontrar el síndrome de un código autocorrector (criptografía basada en códigos).

5.1 Criptografía basada en funciones hash

5.1.1 Introducción a las funciones hash

Como ya hemos comentado, este tipo de sistemas utlizan un tipo de función especial denominada *hash*.

Definición 5.1. Una *familia de funciones hash* es una 4-tupla $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$, donde:

1. $\mathcal{X}$ es un conjunto de posibles *mensajes*.
2. $\mathcal{Y}$ es un conjunto finito de posibles *resúmenes*.
3. $\mathcal{K}$ es el *espacio de claves*, un conjunto finito de posibles claves.
4. Para cada $K \in \mathcal{K}$, existe una *función hash* $h_K \in \mathcal{H}$, con $h_k : \mathcal{X} \to \mathcal{Y}$.

Observemos que, mientras que $\mathcal{X}$ puede ser un conjunto infinito, $\mathcal{Y}$ debe ser finito.

Definición 5.2. Decimos que una pareja $(x, y) \in \mathcal{X} \times \mathcal{Y}$ es una *pareja válida* sobre la clave K si $h_K(x) = y$.

El gran interés criptográfico en las funciones hash subyace en aquellas en las que sólo sea posible (o computacionalmente factible) calcular una pareja válida (x, y) si se conoce x de antemano, calculando $y = h_K(x)$.

La mayor parte de las funciones hash utilizadas no utilizan una clave $K \in \mathcal{K}$ para ser definidas. En la definición anterior, esto se corresponde a que $|\mathcal{K}| = 1$, es decir, la familia tiene una única función hash. En este caso, podemos suprimir la clave en la definición de la función.

Definición 5.3. Una *función hash sin clave* es una función $h : \mathcal{X} \to \mathcal{Y}$, con $\mathcal{X}$ y $\mathcal{Y}$ definidas en la Definición 5.1.

En lo que sigue, trabajaremos con una función hash h sin clave. Como hemos adelantado, nos interesa que h verifique que sólo sea posible calcular una pareja válida (x, y) conociendo x de antemano. Más formalmente, nos interesa que sea difícil resolver estos problemas para h:

Definición 5.4. Sea $h : \mathcal{X} \to \mathcal{Y}$ una función hash sin clave. Se dice que h es *resistente a la preimagen* si es difícil resolver el siguiente problema: dado $y \in \mathcal{Y}$, encontrar $x \in \mathcal{X}$ tal que $h(x) = y$.

Definición 5.5. Sea $h : \mathcal{X} \to \mathcal{Y}$ una función hash sin clave. Se dice que h es *resistente a la segunda preimagen* si es difícil resolver el siguiente problema: dado $x \in \mathcal{X}$, encontrar $x' \in \mathcal{X}$ tal que $x \neq x'$ pero $h(x') = h(x)$.

Definición 5.6. Sea $h : \mathcal{X} \to \mathcal{Y}$ una función hash sin clave. Se dice que h es *resistente a la colisión* si es difícil resolver el siguiente problema: encontrar $x, x' \in X$ con $x \neq x'$ tales que $h(x) = h(x')$.

Es importante observar la sutil diferencia entre la Definición 5.5 y la Definición 5.6: mientras que en la primera tenemos un valor x fijado de antemano, en el segundo no. De hecho, la resistencia a la colisión implica la resistencia a la segunda preimagen, pero el recíproco no es cierto en general [9].

Hay funciones que cumplen con estos criterios, es decir, que no se han encontrado algoritmos que resuelvan ninguno de los tres problemas de forma eficiente. Un buen ejemplo de ello es el *SHA* o *Secure Hash Algorithm*, en sus versiones más modernas (*SHA-256*, *SHA-384*, *SHA-512*) [10]. Para otras versiones más antiguas, sin embargo, como el *SHA-1*, se han encontrado ataques que producen colisiones [11].

5.1.2 Un esquema de firma: el esquema de un solo uso de Lamport-Diffie (LD-OTS)

Presentamos en esta sección el esquema de firma de un solo uso de Lamport-Diffie (LD-OTS). La seguridad de este esquema depende de la propiedad de resistencia a la segunda preimagen de la función hash utilizada. Además, necesita una función resistente a la preimagen. En caso de que la función hash utilizada posea también esta propiedad, es posible usar la misma función en las dos partes del algoritmo.

Esquema de firma de Lamport-Diffie

Parámetros: Un número natural n, el número de bits de la función hash usada, y dos funciones $f : \{0,1\}^n \to \{0,1\}^n$ y $H : \{0,1\}^* \to \{0,1\}^n$, siendo f una función resistente a la preimagen y H una función hash.

Generación de claves: Se elige una clave privada X con $2n$ números de n bits elegidos aleatoriamente de manera uniforme, de manera que

$$X = (x_{n-1}[0], x_{n-1}[1], \cdots, x_1[0], x_1[1], x_0[0], x_0[1])$$

La clave pública Y se calcula evaluando cada coordenada mediante f, es decir,

$$Y = (y_{n-1}[0], y_{n-1}[1], \cdots, y_1[0], y_1[1], y_0[0], y_0[1])$$

donde $y_i[j] = f(x_i[j])$, $0 \leq i \leq n-1$, $j = 0, 1$.

Proceso de firma: Para firmar un documento $M \in \{0,1\}^*$, se calcula $H(M) = (d_{n-1}, \ldots, d_1, d_0)$, y la firma del documento es

$$\sigma = (x_{n-1}[d_{n-1}], \ldots, x_1[d_1], x_0[d_0])$$

Proceso de verificación: Para verificar un mensaje firmado (M, σ) con una clave pública Y, se calcula $H(M) = (d_{n-1}, \ldots, d_1, d_0)$ y se comprueba que para cada $j = 0, \ldots, n-1$:

$$f(\sigma_j) = y_j[d_j]$$

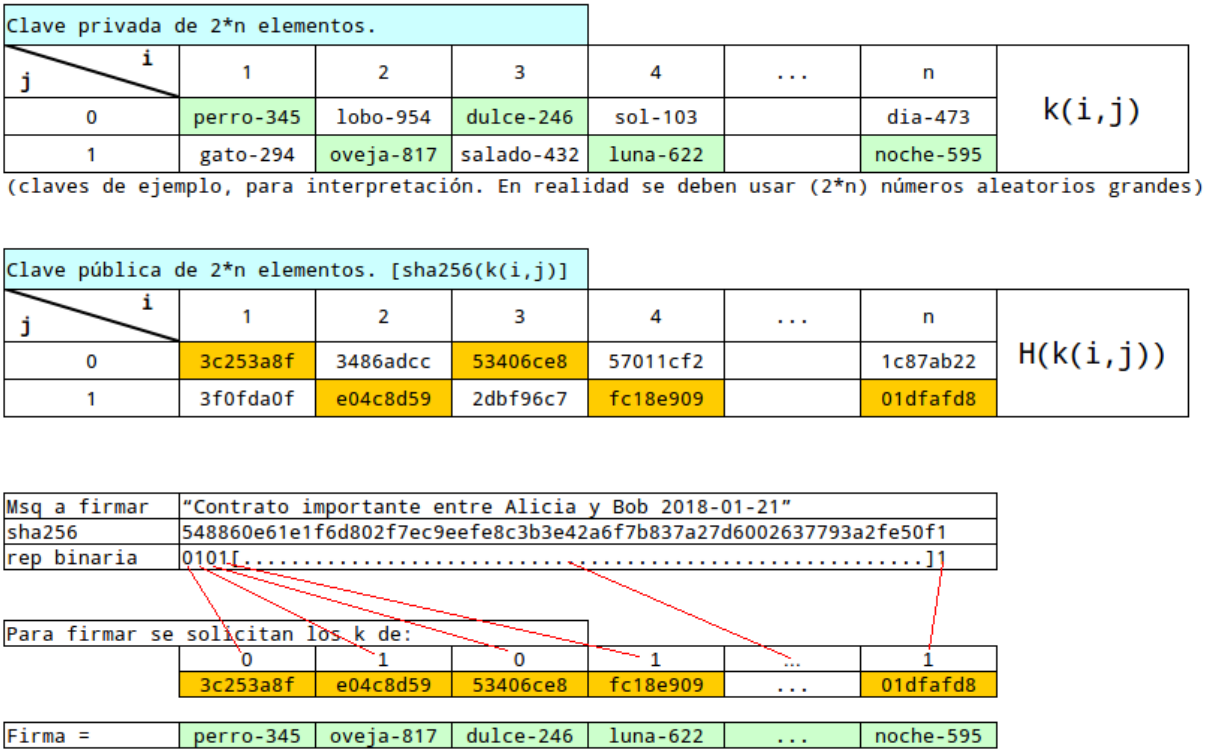


Figura 5.1: Esquema del proceso de firma en el esquema LD-OTS. La verificación se hace calculando la función hash del mensaje firmado, y comprobando que el hash de cada elemento de la firma coincide con la clave pública en cada bit del hash del mensaje firmado. Imagen de Ignacio Zelaya, bajo licencia Creative Commons Attribution-Share Alike 4.0 International.

La seguridad de este esquema radica en la resistencia a la preimagen y las características de la función hash utilizada. En efecto, un atacante Óscar quisiera firmar un documento M' distinto a M, debería proceder de la siguiente manera:

1. Calcular $H(M') = (d'_{n-1}, d'_{n-2}, \ldots, d'_1, d'_0)$.

2. Hallar unos elementos x_j, $j = 0, \ldots, n-1$ tales que

$$f(x_j) = y_j[d'_j]$$

3. Firmar con $\sigma = (x_j)_{j=0,\ldots,n-1}$.

Para aquellos j en los que $d_j = d'_j$, Óscar puede elegir $x_j := \sigma_j$ de la firma del documento M que Alice proporcionó. Pero para los otros casos, la propiedad de resistencia a la preimagen de f hace que para Óscar sea computacionalmente infactible hallar dichos elementos. Por tanto, su única opción sería encontrar M' tal que $H(M) = H(M')$, de manera que $d_j = d'_j$ para todo $j = 0, \ldots, n-1$. Pero la propiedad de resistencia a la segunda preimagen de H hace que hallar dicho M' sea computacionalmente infactible para Óscar.[1]

Mediante este ejemplo es sencillo ver por qué este esquema solo debe usarse una vez para cada par de claves. Si Alice emitiera dos documentos firmados (M, σ) y (M', σ') distintos con el mismo par de claves privada y pública (X, Y), entonces se harían públicos

$$\sigma = (x_{n-1}[d_{n-1}], \ldots, x_1[d_1], x_0[d_0])$$

$$\sigma' = (x_{n-1}[d'_{n-1}], \ldots, x_1[d'_1], x_0[d'_0])$$

Entonces, ahora Óscar podría generar firmas de documentos cuyos hashes fuesen combinaciones de los hashes de M y M'; por ejemplo M'' tal que

$$H(M'') = (d_{n-1}, d'_{n-2}, \ldots, d'_3, d_2, d'_1, d_0)$$

De hecho, si resultase que $d_j \neq d'_j$ para todo j, Óscar conocería la clave privada X completa y podría firmar cualquier documento.

[1] Estas ideas intuitivas han de probarse formalmente. En la última parte de esta sección, probaremos que si existe un ataque contra este sistema, entonces podremos construir un algoritmo que encuentre preimágenes de f. Por tanto, este sistema será seguro si la función f subyacente es segura.

5.1.3 Experimento computacional: rompiendo el sistema LD-OTS con 2 firmas

En este experimento, veremos cómo se puede atacar el sistema si Alice, en contra del protocolo, firma dos documentos diferentes con el mismo par de claves (X, Y). Haremos una comparación entre un ataque de fuerza bruta y un ataque específico cuando se tienen dos firmas con una misma clave.

Supongamos, en primer lugar, que un atacante Óscar recibe un documento firmado (M, σ) de Alice con una clave pública Y, y quiere emitir otro documento firmado (M', σ') usando su clave pública. El algoritmo de búsqueda por fuerza bruta que ejecutaría Óscar sería el siguiente:

Algoritmo 5.1. Búsqueda por fuerza bruta

Entrada: Un conjunto de documentos posibles S, una clave pública X y un documento firmado (M, σ) tal que $M \notin S$.
Salida: Un documento firmado (M', σ') con $M' \in S$, o *error*.

1. Calcular $d := H(M)$.
2. Para cada $M' \in S$ hacer:
   (a) Calcular $d' := H(M')$.
   (b) **Si** $d = d'$, **entonces devolver** (M', σ).
3. **Devolver** *error*.

Evidentemente, cuanto más grande sea el conjunto S de mensajes posibles, más probable es que el algoritmo dé con un documento firmado. Por contrario, cuanto más grande sea el parámetro n del sistema de firma, menos probable es dar con un elemento cuyo hash coincida con el documento firmado original.

En la siguiente tabla se recogen la cantidad de documentos probados antes de dar con un documento firmado con la misma firma que el original para cada valor de n. Los detalles de la implementación en Python se pueden consultar en el último apéndice de este trabajo; para las funciones f y H se ha usado la función *SHA-256* truncada a los primeros n bits deseados en cada caso.

n	Número de documentos	Tiempo
8	64	0,0021 segundos
16	10,350	0,1206 segundos
24[1]	$>$ 30,233,088	$>$ 6,1 minutos
32[1]	$>$ 30,233,088	$>$ 6,1 minutos

[1] El algoritmo no fue capaz de producir un documento firmado falsificado.

Como podemos ver, la complejidad del problema escala extremadamente rápido conforme aumenta el valor de n. Sin embargo, si Alice firmase dos documentos distintos, entonces Óscar podría abordar este problema de manera ligeramente distinta, pues no solo le valdría encontrar un documento cuyo hash sea el de alguno de los dos documentos que ha firmado Alice; podría hacer combinaciones de ellos.

Algoritmo 5.2. Búsqueda con dos documentos firmados

Entrada: Un conjunto de documentos posibles S, una clave pública X y dos documentos firmados $(M, \sigma), (M', \sigma')$ con $M \neq M'$, tales que $M, M' \notin S$.
Salida: Un documento firmado (M'', σ'') con $M'' \in S$, o *error*.

1. Calcular $d := H(M)$ y $d' := H(M')$.
2. Para cada $M'' \in S$ hacer:
   (a) Calcular $d'' := H(M')$.
   (b) Poner $\sigma'' := ()$ (una lista vacía).
   (c) Para cada $j = 0, \ldots, n-1$ hacer:
      i. **Si** $d''_j = d_j$, **entonces** poner $\sigma''_j = \sigma_j$.
      ii. **Si no, si** $d''_j = d'_j$, **entonces** poner $\sigma''_j = \sigma'_j$.
      iii. **Si no**, **volver al paso 2**.
   (d) **Devolver** (M'', σ'').
3. **Devolver** *error*.

En la siguiente tabla se recogen los resultados de aplicar este algoritmo para distintos valores de n, con las mismas claves que se usaron en el caso anterior.

n	Número de documentos	Tiempo
8	17	0,000986 segundos
16	1,679	0,0229 segundos
24	12,071	0,1556 segundos
32	38,320	0,5392 segundos

Como vemos, este ataque comparado con la búsqueda por fuerza bruta es sumamente efectivo. Por ello, se dice que esta firma es de *un solo uso*.

5.1.4 Seguridad del esquema LD-OTS

En esta sección, como ya adelantamos, veremos que el esquema de firma LD-OTS es, al menos, tan seguro como la función resistente a la preimagen utilizada: podremos dar un resultado de irrompibilidad del sistema LD-OTS suponiendo que la función f del esquema cumple ciertas propiedades. Para ello, necesitamos precisar el concepto de *resistencia a la preimagen* de una función hash un poco más [9].

Definición 5.7. Sea S un conjunto. Denotamos $x \xleftarrow{\$} S$ a un elemento $x \in S$ escogido aleatoriamente con una distribución uniforme.

Por contrario, dado $x \in S$, denotamos $y \leftarrow x$ al proceso de definir $y := x$.

Definición 5.8. Un *adversario*, denotado ADV, es cualquier algoritmo probabilístico con cualquier número de entradas.

Por ejemplo, los Algoritmos 5.1 y 5.3 presentados en la sección anterior son ejemplos de adversarios, que podríamos haber denotado ADV_{Bruta} y $\text{ADV}_{2documentos}$, por ejemplo.

Definición 5.9. Sea $\mathcal{G} = (\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$ una familia de funciones hash, y sean $t, \varepsilon > 0$. Se dice que la familia $\mathcal{G}$ es (t, ε) *resistente a la preimagen* si para todo adversario

$$\text{ADV} : \mathcal{K} \times \mathcal{Y} \to \mathcal{X} \cup \{error\}$$

que dado $K \in \mathcal{K}, y = g_k(x)$ para cierto $x \in \mathcal{X}$, devuelve $x' \in \mathcal{X}$ tal que $h_K(x') = y$ o *error* y que puede ejecutarse en tiempo t, verifica que

$$Pr\left[k \xleftarrow{\$} \mathcal{K}, x \xleftarrow{\$} \mathcal{X}, y \leftarrow g_k(x), x' \xleftarrow{\$} \text{ADV}(k, y) : g_k(x') = y\right] \leq \varepsilon$$

De manera similar, podrían extenderse las Definiciones 5.5 y 5.6. Ahora, definimos que un sistema de firma sea *irrompible por un ataque de mensajes elegidos adaptativamente*.

Definición 5.10. Sea $S = (\mathcal{P}, \mathcal{A}, \mathcal{K}, \mathcal{S}, \mathcal{V})$ un sistema de firma y (s_k, p_k) las claves pública y privada asociadas a una clave $k \in \mathcal{K}$. Definimos un *oráculo de firma* $\mathcal{O}(sk, \cdot) : \mathcal{P} \to \mathcal{A}$ como una función que dado un documento $x \in \mathcal{P}$, devuelve su firma $\sigma \in \mathcal{A}$.

Teóricamente, oráculo de firma hace la misma función que el algoritmo de firma del sistema. Sin embargo, a diferencia del segundo, un atacante tiene acceso a este oráculo, aun sin conocer la clave privada, y que a diferencia del algoritmo de firma, su uso puede tener ciertas limitaciones (por ejemplo, en lo que sigue, un atacante sólo podrá usar este oráculo un número limitado de veces).

Definición 5.11. Sea $S = (\mathcal{P}, \mathcal{A}, \mathcal{K}, \mathcal{S}, \mathcal{V})$ un sistema de firma, (s_k, p_k) las claves pública y privada asociadas a una clave $k \in \mathcal{K}$ y $\mathcal{O}(sk, \cdot)$ un oráculo de firma del sistema. Definimos un *falsificador* $\text{FOR}^{\mathcal{O}(sk,\cdot)}(pk)$ a un algoritmo probabilístico que, después de hacer como máximo $q \in \mathbb{N}$ consultas al oráculo, es decir, después de obtener p documentos firmados de la forma

$$\{(M_1, \sigma_1), (M_2, \sigma_2), \cdots, (M_p, \sigma_p)\}, p \leq q$$

encuentra un documento firmado (M', σ') con $M' \neq M_i$ para todo $i = 1, \ldots, p$, o bien devuelve *error*. La elección de los mensajes que se firman con el oráculo puede depender de las firmas anteriores, es decir, los mensajes pueden ser elegidos adaptativamente.

El oráculo hace referencia en realidad a los documentos que ya han sido firmados por el usuario legítimo del sistema; por ello, en el sistema LD-OTS tenemos que $q = 1$, porque sólo debe firmarse un elemento con cada par de claves.

Definición 5.12. Sea S un sistema de firma y $t, \varepsilon > 0$, $q \in \mathbb{N}$. Se dice que S es (t, ε, q) *existencialmente irrompible por un ataque de mensajes elegidos adaptativamente*, o simplemente, es un (t, ε, q) *sistema de firma*, si para cualquier falsificador FOR con q elegido anteriormente y que es capaz de ejecutarse en tiempo t, la probabilidad de que FOR se ejecute con éxito (encuentre el mensaje firmado) es menor o igual que ε.

Con estas definiciones, ya podemos dar el resultado que asegura que el sistema LD-OTS es seguro si la función hash f utilizada es resistente a la preimagen.

Teorema 5.1. *Sea $n \in \mathbb{N}$, $t_{OW}, \varepsilon_{OW} \geq 0$ y $\mathcal{F} = (\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$ una familia de funciones hash $(t_{OW}, \varepsilon_{OW})$ resistentes a la preimagen. Entonces, la firma LD-OTS con $f \in \mathcal{F}$ es un $(t_{OTS}, \varepsilon_{OTS}, 1)$ sistema de firma con $\varepsilon_{OTS} = 4n \cdot \varepsilon_{OW}$ y $t_{OTS} = t_{OW} - t_{SIG} - t_{GEN}$, donde t_{SIG} y t_{GEN} son los tiempos de firma y generación de claves del sistema, respectivamente..*

Demostración. Para demostrar el teorema, procederemos por reducción al absurdo. Sea $\text{FOR}^{\mathcal{O}(X,\cdot)}$ un falsificador de LD-OTS, que se ejecuta en tiempo $t = t_{OW} - t_{SIG} - t_{GEN}$ pero con probabilidad de éxito $\varepsilon > \varepsilon_{OTS} = 4n \cdot \varepsilon_{OW}$. Construiremos un adversario ADV_{Pre} que encuentra preimágenes de funciones $f \in F$. Es decir,

$$\text{ADV}_{Pre} : \mathcal{K} \times \mathcal{Y} \to \mathcal{X} \cup \{error\}$$

tal que dado una clave $k \in \mathcal{K}$ y una imagen $y = f_k(x) \in \mathcal{Y}$ para algún $x \in \mathcal{X}$ y $f_k \in \mathcal{H}$, devuelve $x' \in \mathcal{X}$ tal que $f_k(x') = y$, o *error*. Este adversario actuaría de la siguiente manera.

1. Genera un par de claves pública y privada (X, Y) del sistema LD-OTS, con $Y = (y_{n-1}[0], y_{n-1}[1], \ldots, y_0[0], y_0[1])$.
2. Después, elige dos índices $a \in \{0, \ldots, n-1\}$ y $b \in \{0, 1\}$ de manera aleatoria; y remplaza $y_a[b]$ en la clave pública Y con la imagen objetivo del algoritmo del que queremos hallar la preimagen, y.

3. El adversario ahora hará el rol de oráculo, y ejecuta $\text{FOR}^{\mathcal{O}(X,\cdot)}$. Ya que $q = 1$, este falsificador puede hacer una consulta al oráculo, o ninguna. Si la hace, pedirá firmar un documento $H(M) = (m_{n-1}, \ldots, m_0)$. Si $m_a = 1 - b$, entonces el adversario puede firmar usando la clave privada X, ya que $f_k(x_a[1-b]) = y_a[1-b]$ al no haberse modificado $y_a[1-b]$ (sino $y_a[b]$). En caso contrario, devuelve *error*, lo que hace que el falsificador no pueda continuar (en este caso, el algoritmo completo fallaría).

4. Si el falsificador se ejecuta con éxito, devolverá un mensaje $M' = (m'_{n-1}, \cdots, m'_0)$ firmado con $\sigma' = (\sigma'_{n-1}, \cdots, \sigma'_0)$. Si $m'_a = b$, entonces la preimagen buscada es σ'_a, ya que $f_k(\sigma'_a) = y_a[b] = y$ por definición de documento firmado.

Algoritmo 5.3. ADV_{PRE}

Entrada: $k \xleftarrow{\$} \mathcal{K}$ e $y \in \mathcal{Y}$.
Salida: $x \in \mathcal{X}$ tal que $y = f_k(x)$, o *error*.

1. Generar un par de claves (X, Y) del sistema LD-OTS.
2. Escoger $a \xleftarrow{\$} \{0, \ldots, n-1\}$ y $b \xleftarrow{\$} \{0, 1\}$.
3. Reemplazar $y_a[b]$ por y en la clave privada Y.
4. Ejecutar $\text{FOR}^{\mathcal{O}(X,\cdot)}(Y)$.
5. **Si** $\text{FOR}^{\mathcal{O}(X,\cdot)}(Y)$ consulta al oráculo con $M = (m_{n-1}, \ldots, m_0)$, **entonces:**
   (a) **Si** $m_a = (1-b)$, **entonces** firmar M usando X y responder a $\text{FOR}^{\mathcal{O}(X,\cdot)}(Y)$ con la firma σ.
   (b) **Si no**, **devolver** *error*.
6. **Si** $\text{FOR}^{\mathcal{O}(X,\cdot)}(Y)$ devuelve un documento firmado (M', σ'), $M' = (m'_{n-1}, \ldots, m'_0)$, $\sigma' = (\sigma'_{n-1}, \ldots, \sigma'_0)$, **entonces:**
   (a) **Si** $m'_a = b$, **entonces devolver** σ'_a.
   (b) **Si no**, **devolver** *error*.
7. **Si no**, **devolver** *error*.

Veamos ahora el tiempo que toma en ejecutarse t_{OW} y la probabilidad de éxito ε_{OW} de este adversario. Denotando t_{GEN} el tiempo de generación de claves del esquema LD-OTS, t_{SIG} el tiempo de firma y t, ε el tiempo que toma en ejecutarse el falsificador $\text{FOR}^{\mathcal{O}(X,\cdot)}(Y)$ y su probabilidad de éxito respectivamente, entonces

$$t_{ADV} = t + t_{GEN} + t_{SIG} \tag{5.1}$$

Por otro lado, observemos que para que el adversario tenga éxito, deben ocurrir tres cosas:

- Si el falsificador pregunta al oráculo, que $m_a = (1 - b)$. Ya que b se elige de manera aleatoria entre 0 y 1 con una distribución uniforme, la probabilidad de que esto ocurra es exactamente $\frac{1}{2}$.

- Que el falsificador tenga éxito en devolver un documento firmado. La probabilidad que esto ocurra es ε.

- Que $m'_a = b = 1 - m_a$. Ya que $M' \neq M$, existe $c \in \{0, \ldots, n-1\}$ tal que $m'_c = 1 - m_c$. El adversario tiene éxito si $c = a$, lo que ocurre con probabilidad, al menos, $\frac{1}{2n}$.

Por tanto, tenemos que

$$\varepsilon_{ADV} \geq \frac{1}{2} \cdot \varepsilon \cdot \frac{1}{2n} = \frac{\varepsilon}{4n} \tag{5.2}$$

Sin embargo, ya que por hipótesis $t = t_{OW} - t_{SIG} - t_{GEN}$, de la ecuación (5.1) deducimos que $t_{ADV} = t_{OW}$, y como $\varepsilon > 4n \cdot \varepsilon_{OW}$, de la ecuación (5.2) deducimos que:

$$\varepsilon_{ADV} \geq \frac{\varepsilon}{4n} > \varepsilon_{OTS}$$

Hemos encontrado un adversario que se ejecuta en tiempo t_{OW} y con probabilidad de éxito mayor que ε_{OTS}. Esto contradice el hecho que $\mathcal{F}$ sea una familia de funciones hash $(t_{OW}, \varepsilon_{OW})$ resistente a la preimagen, lo que completa la demostración. □

Este hecho prueba que toda la seguridad de este equema, se traslada a la seguridad de la función hash utilizada subyacente. Ya que no se han encontrado ataques efectivos a funciones hash comúnmente utilizadas como la familia *SHA-256*, ni en ordenadores clásicos ni cuánticos, este esquema se propone como una alternativa a otros sistemas de firma basados en la factorización de los números enteros, como el *RSA*.

5.2 Criptografía basada en códigos

A diferencia de la sección anterior, de la criptografía basada en códigos tenemos ejemplos de criptosistemas (en vez de sistemas de firma). Se basan en el problema de la decodificación de una palabra de tipo de códigos autocorrectores espciales, los *códigos Goppa*.

5.2.1 Introducción a los códigos Goppa

Antes de presentar los códigos Goppa y sus propiedades, damos unas pequeñas defincione sobre códigos algebraicos.

Definición 5.13. Sea $\mathbb{F}_q$ un cuerpo finito de q elementos. Un *código* es un subconjunto no vacío $\mathcal{C} \subset \mathbb{F}_q^n$. En este caso, decimos que $\mathcal{C}$ tiene *longitud* n. Sus elementos se denominan *palabras* del código.

Si además $\mathcal{C}$ es un subespacio vectorial de $\mathbb{F}_q^n$ de dimensión $0 \leq k \leq n$, entonces decimos que es un $[n, k]$-código lineal sobre $\mathbb{F}_q$.

Definición 5.14. Sea $\mathcal{C}$ un $[n, k]$-código lineal sobre $\mathbb{F}_q$. Una *matriz generadora* de $\mathcal{C}$ es una matriz $G \in M_{k\times n}(\mathbb{F}_q)$ cuyas filas forman una base algebraica de $\mathcal{C}$.

El *código dual* $\mathcal{C}^\perp$ de $\mathcal{C}$ es el subespacio ortogonal de $\mathcal{C}$ sobre el producto escalar usual de $\mathbb{F}_q$. Es, por tanto, un $[n, n-k]$-código lineal sobre $\mathbb{F}_q$.

Llamamos a una matriz $H \in M_{(n-k)\times n}(\mathbb{F}_q)$ *matriz de control de paridad* de $\mathcal{C}$ a una matriz generadora de $\mathcal{C}^\perp$.

Definición 5.15. Sean $x, y \in \mathbb{F}_q^n$. Denominamos *distancia de Hamming* entre x e y al número de coordenadas que difieren entre x e y, es decir:

$$d(x, y) = \#\{i : x_i \neq y_i, i = 1, \ldots, n\}$$

Dado un $[n, k]$-código lineal $\mathcal{C} \subset \mathbb{F}_q^n$, llamamos *distancia mínima de* $\mathcal{C}$, y lo denotamos $d(\mathcal{C})$, a la menor distancia de Hamming entre dos palabras distintas cualesquiera de $\mathcal{C}$, es decir;

$$d(\mathcal{C}) = \min_{\substack{x,y\in\mathcal{C}\\ x\neq y}} \{d(x, y)\}$$

Un $[n, k]$-código lineal con distancia mínima d lo denotamos un $[n, k, d]$-código lineal.

Es sencillo ver, además, que la distancia de Hamming es una métrica en $\mathbb{F}_q^n$.

Definición 5.16. Sea $\mathcal{C}$ un código de $\mathbb{F}_q^n$. Un *decodificador* es una aplicación

$$D_{\mathcal{C}} : \mathbb{F}_q^n \to \mathcal{C}$$

Se dice que $D_{\mathcal{C}}$ corrige t errores si para cada $e \in \mathbb{F}_q^n$ y todo $x \in \mathcal{C}$, se verifica que si $wt(e) \leq t$, entonces $D_c(x+e) = x$, donde $wt(x) = \#\{i : x_i \neq 0, i = 0, \ldots, n-1\}$.

Al valor $wt(x)$ se le *peso* de la palabra $x \in \mathbb{F}_q^n$, y al vector $e \in \mathbb{F}_q^n$ se denomina *vector error*.

Finalmente, podemos dar una definción de un código Goppa. Aunque existen diversas definiciones, nosotros seguiremos [12].

Definición 5.17. Sea $g(z)$ un polinomio mónico de grado t sobre $\mathbb{F}_{q^m}$. Sea $L = \{\gamma_0, \gamma_1, \ldots, \gamma_{n-1}\} \subset \mathbb{F}_{q^m}$, tal que $g(\gamma_i) \neq 0$ para todo $0 \leq i \leq n-1$. Definimos el *código Goppa* $\Gamma(L, g)$ con el *polinomio de Goppa* $g(z)$ como el conjunto de palabras $c = (c_0, c_1, \ldots, c_{n-1}) \in \mathbb{F}_q^n$ tales que

$$\sum_{i=0}^{n-1} \frac{c_i}{z - \gamma_i} \equiv 0 \mod g(z) \tag{5.3}$$

Si el polinomio de Goppa $g(z)$ es además irreducible, entonces se dice que $\Gamma(L, g)$ es un *código de Goppa irreducible*.

Con esta definición, es fácil ver que un código de Goppa es lineal viendo que la suma de palabras y el producto de escalares por palabras verifica la ecuación (5.3). Así, podemos dar el siguiente resultado, cuya demostración puede consultarse en [12]:

Proposición 5.1. *Tomando* $h_j := g(\gamma_j) - 1$, $j = 0, \ldots, n-1$, *una matriz de control de paridad de* $\Gamma(L, g)$ *es:*

$$\begin{pmatrix} h_0 & h_1 & \cdots & h_{n-1} \\ h_0\gamma_0 & h_1\gamma_1 & \cdots & h_{n-1}\gamma_{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ h_0\gamma_0^{t-1} & h_1\gamma_1^{t-1} & \cdots & h_{n-1}\gamma_{n-1}^{t-1} \end{pmatrix}$$

Sin exigir restricciones al código, tememos las siguientes cotas para la dimensión y distancia mínima de un código de Goppa (véase [12] para una demostración a partir de otro tipo de códigos, los códigos BCH.)

Proposición 5.2. *Con la notación anterior, un código de Goppa* $\Gamma(L, g)$ *tiene dimensión* $k \geq n - mt$ *y distancia mínima* $d \geq t + 1$.

Sin embargo, exigiendo $q = 2$ y que el polinomio de Goppa g no tenga raíces múltiples (por ejemplo, si es irreducible), se puede demostrar el siguiente resultado ([12]):

Proposición 5.3. *Sea* $q = 2$, *y* $g(z) \in \mathbb{F}_{2^m}[X]$ *un polinomio mónico de grado* t *sin raíces múltiples. Entonces,* $\Gamma(L, g)$ *tiene distancia mínima* $d \geq 2t + 1$.

La utilidad de los códigos de Goppa en criptografía radica en que existen algoritmos eficientes para decodificar palabras correctamente con un número de errores hasta la mitad de la distancia mínima de $\Gamma(L, g)$, con $\mathcal{O}(n \cdot t \cdot m^2)$ operaciones binarias ([13]).

5.2.2 Un criptosistema: el sistema de McEliece

El criptosistema de clave pública de McEliece fue presentado en 1978 ([14]), y en la actualidad no se han encontrado ataques efectivos contra este sistema (aunque sus parámetros de seguridad deben ser adaptados con la llegada de los ordenadores cuánticos, [6]). Presentamos ahora la descripción del sistema.

Criptosistema de clave pública de McEliece

Parámetros: Dos números naturales n y t, con $t << n$ (mucho menor que n).

Generación de claves: Se calcula la matriz generadora G de un código de Goppa $\mathcal{G}$ irreducible sobre $\mathbb{F}_2^n$ de dimensión k, con un polinomio de Goppa de grado t. Por tanto, la distancia mínima de este código es $d \geq 2t+1$ (Proposición 5.3). A continuación, se eligen dos matrices aleatorias

- $S \in M_{k\times k}(\mathbb{F}_2)$, siendo S no singular (invertible).
- P, una matriz $n \times n$ de permutación de dimensión $n \times n$.

Se calcula $G^{pub} = SGP$, y la clave pública es (G^{pub}, t).

La clave privada es $(S, D_{\mathcal{G}}, P)$, donde $D_{\mathcal{G}}$ es un algoritmo de decodificación eficiente del código Goppa $\mathcal{G}$ que corrige hasta t errores (ver sección anterior, [13]).

Encriptado: Para encriptar un texto plano $m \in \mathbb{F}_2^k$, se elige una clave efímera que consiste en un vector $z \in \mathbb{F}_2^n$ de peso t. El texto cifrado $c \in \mathbb{F}_2^n$ se calcula como

$$c = mG^{pub} + z$$

Desencriptado: Para desencriptar un texto cifrado $c \in \mathbb{F}_2^n$, se calcula cP^{-1}. Seguidamente, se utiliza el algoritmo de decodificación para hallar $m' := D_{\mathcal{G}}(cP^{-1})$. Posteriormente, se hallan las coordenadas de m' respecto de la base dada por las filas de G, m'', y finalmente, se recupera el texto plano hallando $m''S^{-1}$.

Si el texto a encriptar fuese de longitud mayor que k, basta con dividir el mensaje en bloques de longitud k, y encriptarlos por separado. La desencriptación sigue un proceso análogo, desencriptando por bloques de longitud n y recuperando el texto plano original. En el siguiente teorema vemos que, efectivamente, el sistema está bien definido. Para ello, debemos verificar que el criptosistema verifica la propiedad 4 de la Definición 2.1 (en este caso, $\mathcal{P} = \mathbb{F}_q^k$, $\mathcal{C} = \mathbb{F}_q^n$, $\mathcal{K}$ sería el conjunto de todos los códigos Goppa posibles de $\mathbb{F}_q^n$).

Teorema 5.2. *En el criptosistema de clave pública de McEliece, el proceso de desencriptado es inverso al de encriptado, es decir, verifica la propiedad 4 de la Definición 2.1.*

Demostración. Sea $c \in \mathbb{F}_2^n$ un mensaje encriptado usando el algoritmo de encriptado del criptosistema de clave pública de McEliece. Por tanto,

$$c = mG^{pub} + z = mSGP + z$$

Hallando cP^{-1} (que existe al ser P una matriz de permutación, luego $\det P = \pm 1$), obtenemos:

$$cP^{-1} = mSG + zP^{-1}$$

Al ser G la matriz generadora del código $\mathcal{G}$, $mSG \in \mathcal{G}$, por ser un vector formado por una combinación de filas de G (que forman una base del código). Por otro lado, $wt(zP^{-1}) = wt(z) = t$, por ser zP^{-1} un vector con las mismas coordenadas que z, pero permutadas. Ya que el algoritmo $D_{\mathcal{G}}$ corrige hasta t errores, entonces (ver Definición 5.16):

$$D_{\mathcal{G}}(cP^{-1}) = D_{\mathcal{G}}(mSG + zP^{-1}) = mSG$$

Las coordenadas de $m' := mSG$ respecto de la base del código son $m'' := mS$. Finalmente,

$$m''S^{-1} = mSS^{-1} = m$$

que es el texto plano original. □

La seguridad de este algoritmo se apoya en que un atacante Óscar que dado $c \in \mathbb{F}_q^n$ quisiera obtener el texto plano $m \in \mathbb{F}_q^k$ sin conocer la clave privada, tiene dos opciones: o bien recuperar la matriz generadora del código original G a partir de G^{pub}, con lo que podría aplicar el algoritmo de decodificación D_G, o bien recuperar el texto plano original sin usar el algoritmo de decodificación [14].

El primer ataque, con valores suficientemente grandes de k y n, parece infactible dadas las posibilidades para elegir las matrices P y S. El segundo se conoce como el problema de la decodificación para códigos lineales, y no se han encontrado algoritmos efectivos clásicos ni cuánticos para resolverlo. Aunque el algoritmo de Grover puede ser aplicado a este sistema, basta con reemplazar el parámetro de seguridad n por $(2 + o(1))n$ para protegerse de estos ataques [6].

CAPITULO

6

Conclusiones

APPENDIX

A

Código de los programas

A.1 Implementación de la firma LD-OTS en Python

```
class LDOTS:
    # Inicializa la clase. Especificar el numero de bits que devuelve la funcion hash y la funcion de ida (n), la funcion
    # hash y la funcion de ida correspondientes y si queremos verificar usando una clave publica, se pasa como ultimo parametro.
    def __init__(self, n, hash_f, ow_f, pubK = None):
        self.n = n
        self.hash_f = hash_f
        self.ow_f = ow_f

        if pubK:
            self.pubK = pubK
            self.privK = None

    # Genera un par de claves aleatorias criptograficamente seguras
    def generate_keys(self):
        self.privK = [ ( secrets.randbits(self.n).to_bytes(self.n//8,'big'),secrets.randbits(self.n).to_bytes(self.n//8,'big')) for k in range(self.n) ]
        self.pubK = [ (self.ow_f(k[0]),self.ow_f(k[1])) for k in self.privK ]

    # Exporta las claves en los dos ficheros indicados
    def export_keys(self, pubName, privName):
        pubFile = open(pubName+".PUK","wb")
        pubFile.write(pickle.dumps(self.pubK))
```

```
22          pubFile.close()
23
24          privFile = open(privName+".PRK","wb")
25          privFile.write(pickle.dumps(self.privK))
26          privFile.close()
27
28      # Importa la clave publica del fichero especificado
29      def import_pubK(self,name):
30          file = open(name,"rb")
31          self.pubK = pickle.loads(file.read())
32          file.close()
33
34      # Importa la clave privada del fichero especificado
35      def import_privK(self,name):
36          file = open(name,"rb")
37          self.privK = pickle.loads(file.read())
38          file.close()
39
40      # Importa las dos claves del fichero especificado
41      def import_keys(self,public,private):
42          self.import_pubK(public)
43          self.import_privK(private)
44
45      # Firma el mensaje especificado. El valor devuelto puede pasarse
        directamente al metodo verify(). Es necesario contar con la clave
        privada para usar este metodo.
46      def sign(self,msg):
47          if self.privK:
48              d = bytestobits(self.hash_f(msg))
49              return [ self.privK[j][d[j]] for j in range(self.n) ]
50          else:
51              raise Exception("No hay clave privada; use generate_keys()
        para generar un par de claves!")
52
53      # Devuelve True si el mensaje es autentico, False si no lo es. Es
        necesario haber inicializado esta clase con la clave publica del
        emisor.
54      def verify(self,msg,signature):
55          if self.pubK:
56              d = bytestobits(self.hash_f(msg))
57
58              for j in range(self.n):
59                  if self.ow_f(signature[j]) != self.pubK[j][d[j]]: return
        False
60
61              return True
62          else:
63              raise Exception("No hay clave publica!")
```

A.2 Algoritmo de búsqueda por fuerza bruta

```
start = time.time() # Para medir el tiempo de ejecucion

k=1
for msg in msgGen.iterator(): # msgGen es el generador de mensajes
    digest = lamport.bytestobits(sha256(msg.encode('utf-8')))
    if (digest == dig1):
        print("Documento falsificado!")
        print(msg)
        print("Comprobacion: " + str(l.verify(msg.encode('utf-8'),sig1)))
        break
    k += 1

print(str(k)+" documentos probados.")
print("%s segundos" %(time.time() - start))
```

A.3 Algoritmo de búsqueda con 2 firmas

```
start = time.time() # Para medir el tiempo de ejecucion

k=1
for msg in msgGen.iterator(): # msgGen es el generador de mensajes
    digest = lamport.bytestobits(sha256(msg.encode('utf-8')))
    mysig = []
    for k in range(len(digest)):
        if digest[k]==dig1[k]:
            mysig.append(sig1[k])
        elif digest[k]==dig2[k]:
            mysig.append(sig2[k])
        else:
            break
    if len(mysig)==len(sig1):
        print("Documento falsificado!")
        print(msg)
        print("Comprobacion: " + str(l.verify(msg.encode('utf-8'),mysig))
    )
        break
    cont += 1

print(str(cont)+" documentos probados.")
print("%s segundos" %(time.time() - start))
```

Bibliografía

[1] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, pages 120–126, 1978. 3

[2] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *Advances in Cryptology - CRYPTO '84, LNCS 196*, pages 10–18, 1985. 3

[3] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ECDSA). *International Journal of Information Security*, 1(1):36–63, Aug 2001. 3

[4] Daniel J. Bernstein and Johannes Buchmann And Erik Dahmen, editors. *Post-Quantum Cryptography*. Springer, 2009. 4, 19

[5] Markus Grassl, Brandon Langenberg, Martin Roetteler, and Rainer Steinwandt. Applying Grover's Algorithm to AES: Quantum Resource Estimates. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography*, pages 29–43, Cham, 2016. Springer International Publishing. 4

[6] Daniel J. Bernstein. Grover vs. McEliece. In Nicolas Sendrier, editor, *Post-Quantum Cryptography*, pages 73–80, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. 4, 56, 58

[7] A. Aizpuru Tomás. *Apuntes incompletos de Análisis Funcional*. Servicio de Publicaciones de la Universidad de Cádiz, 2009. 11, 12

[8] Michael A. Nielsen and Isaac L.Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000. 19, 20, 29, 30, 31, 38, 39

[9] Phillip Rogaway and Thomas Shrimpton. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In Bimal Roy and Willi Meier, editors, *Fast Software Encryption*, pages 371–388, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. 43, 49

[10] Henri Gilbert and Helena Handschuh. Security Analysis of SHA-256 and Sisters. In Mitsuru Matsui and Robert J. Zuccherato, editors, *Selected Areas in Cryptography*, pages 175–193, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. 43

[11] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, pages 17–36, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. 43

[12] J. H. van Lint. *Introduction to Coding Theory*. Number 86 in Graduate Texts in Mathematics. Springer-Verlag Berlin Heidelberg, 1999. 55, 56

[13] Nick Patterson. The algebraic decoding of Goppa codes. *IEEE Transactions on Information Theory*, 21(2):203–207, 1975. 56, 57

[14] R. J. McEliece. A public-Key cryptosystem based on algebraic coding theory. *DSN Progress Report*, 42-44:114–116, 1978. 56, 58

[15] Douglas R. Stinson. *Cryptography, Theory and Practice*. Discrete Mathematics And Its Applications. Chapman & Hall/CRC, 3 edition, 2006.

[16] Noson S. Yanofsky and Mirco A. Mannucci. *Quantum computing for computer scientists*. Cambridge University Press, 2008.

[17] Paul R. Halmos. *Finite-Dimensional Vector Spaces*. Springer, 1916.