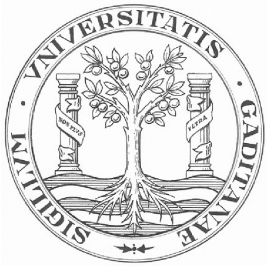


UNIVERSIDAD DE CÁDIZ

FACULTAD DE CIENCIAS

GRADO EN MATEMÁTICAS

CRIPTOGRAFÍA POSCUÁNTICA

Trabajo de fin de grado presentado por

Juan Antonio Guitarte Fernández

Tutores: Dr. Bartolomé López Jiménez
Dr. Juan Ignacio García García

Firma del alumno Firma del tutor

Puerto Real, Cádiz, Julio de 2020

Abstract

In this project, we give a general overview of the current proposals of cryptographic systems that are set to replace the public key systems and digital signature schemes used nowadays. In the first introductory chapter, we give a summary of the principal groups of proposals and some basic notions of cryptography. Then, we study with more detail the fundamentals of quantum computing and how the algorithms that break current systems work. Finally, we give a recap of two groups of schemes that are proposed as alternatives to classical ones: hash-based digital signature schemes, and code-based cryptosystems.

A mi familia en general; pero sobre todo, a mi abuelo.

Resumen

En este trabajo, se da una visión general de las propuestas vigentes en la actualidad de sistemas criptográficos para reemplazar los sistemas de clave pública y de firma digital que se ven afectados por la aparición de los ordenadores cuánticos. En el primer capítulo, introductorio, se ve un resumen de las propuestas y algunas nociones básicas de criptografía. A continuación, se estudian con más profundidad los fundamentos de la computación cuántica y cómo funcionan los algoritmos cuánticos que rompen los sistemas actuales. Finalmente, se da una revisión de dos grandes grupos de sistemas criptográficos que se proponen como alternativas a los esquemas clásicos: aquellos basados en funciones hash, y los construidos sobre códigos autocorrectores.

Agradecimientos

Quiero agradecer, en primer lugar, a todos mis profesores, compañeros y personas que he conocido a lo largo de mi estancia en la universidad. Gracias a vosotros, no solo he conseguido formarme como matemático, sino también como persona. Me llevo muy buenos recuerdos y experiencias de esta etapa de mi vida, y sobre todo, grandes amigos para el futuro.

También quisiera agradecer a mi familia, que son un pilar fundamental de mi vida y que sin ellos no habría podido llegar hasta aquí. Su apoyo en los momentos más difíciles y no tan difíciles han sido cruciales para poder seguir adelante hasta en las circunstancias más adversas.

Como tampoco podía ser de otra manera, quiero agradecer a mis dos tutores, Bartolomé e Ignacio. Las circunstancias han sido duras para todos durante este segundo cuarto de año, sin embargo, gracias a su apoyo y consejo, este trabajo ha sido posible. Muchas gracias.

Juan Antonio Guitarte Fernández

junio 2020

Índice general

CAPITULO

1

Planteamiento y motivación

1.1 El problema de los ordenadores cuánticos

En la actualidad, 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, etc. 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, es 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 (el cual estudiaremos también), 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.

En el resto de este capítulo, veremos qué dos grandes problemas resuelve la criptografía y cómo se usa en la práctica un sistema criptográfico.

1.2 Criptosistemas

El objetivo principal de la criptografía es permitir que dos personas, normalmente referidas como Alice y Bob[1], puedan comunicarse entre ellas a través de un canal inseguro de tal manera que una tercera persona, Óscar, 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". Óscar, 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 1.1. 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 $e_K \in \mathcal{E}$, $d_K \in \mathcal{D}$, con $e_K : \mathcal{P} \to \mathcal{C}$ 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] En la literatura de la criptografía, usar estos personajes ficticios para explicar el funcionamiento de algún sistema o ataque es común. La primera aparición de estos nombres tuvo lugar en el artículo de Rivest, Shamir y Adleman presentando el sistema RSA [1].

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 1.1, 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.

1.3 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 Óscar, 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 Óscar, como sucede en los criptosistemas de clave pública.

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

Definición 1.2. 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*.

El proceso por el cual Alice utilizaría un sistema de firma para firmar es el siguiente:

1. Escoge una clave $K \in \mathcal{K}$ de forma aleatoria, y hace público el algoritmo de verificación ver_K (en la práctica, publica una serie de datos que permiten a cualquier persona determinar la función, normalmente referidos como *clave pública*).

2. Dado un documento $m \in \mathcal{P}$, utiliza el algoritmo de firma sig_K para hallar $y = sig_K(m) \in \mathcal{A}$, y publica el mensaje firmado (m, y).

Si Bob, al recibir el mensaje firmado (m, y), quiere comprobar si realmente ha sido escrito por Alice, solo debe aplicar el algoritmo de verificación ver_K emitido por Alice y comprobar que $ver_K(m, y) = 1$.

Como el algoritmo de firma sig_K no es público y solo lo conoce Alice, un atacante Óscar que quisiera enviar un documento firmado (m', y') a Bob tendría que calcular $y' \in \mathcal{A}$ tal que $ver_K(m', y') = 1$. Si resolver este problema es lo suficientemente difícil, Óscar no podría falsificar ningún documento haciéndose pasar por Alice, lo cual haría al sistema seguro.

CAPITULO

2

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 ([7], 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 ([7]).

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.

2.1 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 2.1. *Sea N un número compuesto de L bits, y sea $x \in \mathbb{Z}/n\mathbb{Z}$ tal que $1 < x < N-1$ una solución de la ecuación*

$$x^2 \equiv 1 \mod N \tag{2.1}$$

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 $x^2 - 1 \equiv 0 \mod N$, 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 $\mathrm{mcd}(x-1, N) > 1$.

Por otro lado, 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 uno de los dos máximos comunes divisores es un factor no trivial de N, puesto que está estrictamente comprendido entre 1 y N. Mediante el algoritmo de Euclides, estos factores pueden calcularse en $O(L^3)$ operaciones (ver [7], pág. 629). □

En general, encontrar una solución de (2.1) 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 $y^r \equiv 1 \mod N$) es par, entonces $y^{r/2}$ es una solución de (2.1); y además cumple las hipótesis de la Proposición 2.1 si $y^{r/2} \not\equiv -1 \equiv N-1 \mod N$ (observemos que no puede ser 1, ya que ello contradiría la definición de r). Si escogemos este número y aleatoriamente entre 2 y $N-2$ (si no es coprimo con N, hemos 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 2.1. Este hecho se recoge en el siguiente teorema.

Teorema 2.1. *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, la probabilidad*

$$P[r \text{ es par y que } x^{r/2} \not\equiv -1 \mod N] \geq 1 - \frac{1}{2^{m-1}}$$

Para demostrarlo, necesitamos primero un lema previo. En lo que sigue, φ denota la función phi de Euler.

Lema 2.1. *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 la probabilidad de que 2^d divida al orden de cualquier elemento de $(\mathbb{Z}/p^\alpha\mathbb{Z})^*$ es $\frac{1}{2}$.*

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 g^k.

- Si es k impar, ya que $\left(g^k\right)^r \equiv g^{kr} \equiv 1 \mod p^\alpha$, por ser $\varphi(p^\alpha)$ el orden de g (es un generador del grupo, luego su orden es el cardinal del grupo), $\varphi(p^\alpha)|kr$. De aquí, 2^d divide a kr, y por ser k impar, necesariamente 2^d divide a r, el orden de g^k.

- Si es k par, entonces

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

  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 $\varphi(p^\alpha)/2$, 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 2.1.

Demostración. Probaremos que

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

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 $x \equiv x_j \mod p_j^{\alpha_j}$. 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[1], de aquí se tiene que r_j es impar, ergo $d_j = 0$ para todo j.

- Si r es par y es $x^{r/2} \equiv -1 \mod N$, entonces $x^{r/2} \equiv -1 \mod p_j^{\alpha_j}$, esto es, $x_j^{r/2} \equiv -1 \mod p_j^{\alpha_j}$. De aquí, $r_j \not| (r/2)$; porque si no, debería ser $x_j^{r/2} \equiv x_j^{r_j k} \equiv (x_j^{r_j})^k \equiv 1^k \equiv 1 \mod p_j^{\alpha_j}$. 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 $x^{r/2} \equiv -1 \mod N$, entonces d_j toma el mismo valor para cada j. En particular, $d_2, \ldots, d_m$ deben ser iguales a d_1, es decir, $2^{d_1} | r_j$ para cada $j = 1, \ldots, m$. Sea ahora 2^{k_j} la mayor potencia de 2 que divide a $\varphi(p_j^{\alpha_j})$. Entonces, como la mayor potencia de 2 que divide a cada r_j es la misma, debe cumplirse que $2^{k_j} | r_j$ si $k_j \leq d_1$ ó $2^{k_j} \not| r_j$ si $k_j > d_1$ para todo $j = 2, \ldots, m$. Por el Lema 2.1, la probabilidad de que esto ocurra es

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

Y por contención de sucesos, entonces

$$P[r \text{ es impar ó } x^{r/2} \equiv -1 \mod N] \leq P[d_j = d_1 \text{ para todo } j = 2, \ldots, m] = \frac{1}{2^{m-1}}$$

Lo cual completa la demostración.

□

Una consecuencia inmediata del Teorema 2.1 es que si el número que queremos factorizar N tiene 2 factores primos (como suele usarse en criptografía, por ejemplo, en el algoritmo RSA), entonces tenemos la garantía de que usando el procedimiento descrito anteriormente, hallaremos al menos un 50 % de las veces un elemento y que nos permita hallar los factores de N usando la Proposición 2.1. Un resumen de un algoritmo para factorizar sería el siguiente:

1. Si N es par, devolver el factor 2.

2. Elegir aleatoriamente un número 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.

[1]Porque $x_j^r \equiv x^r \mod p_j^{\alpha_j}$. Como $x^r \equiv 1 \mod N$, entonces $x^r \equiv 1 \mod p_j^{\alpha_j}$. Luego $x_j^r \equiv 1 \mod p_j^{\alpha_j}$, y de aquí el orden $r_j | r$.

4. Si r es par e $y^{r/2} \not\equiv -1 \mod N$, entonces calcular $\mathrm{mcd}(y^{r/2}-1, N)$ y $\mathrm{mcd}(y^{r/2}+1, N)$, 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.

2.2 Computación cuántica

2.2.1 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 2.1. La puerta lógica AND implementa la operación lógica de conjunción (&) y se define de esta manera:

$$AND : \mathbb{Z}_2^2 \to \mathbb{Z}_2$$

$$(0,0) \longmapsto 0$$

$$(0,1) \longmapsto 0$$

$$(1,0) \longmapsto 0$$

$$(1,1) \longmapsto 1$$

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 2.1. 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[1] $\mathbb{C}^2$, que 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 $x \in \mathbb{C}^2$, $\|x\|^2 = |c_1|^2 + |c_2|^2 = 1^2$. Así, cuando $|c_1|^2 = 1$ y $|c_2|^2 = 0$ 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 $|c_1|^2$ de valer 0 al ser observado, y $|c_2|^2$ de valer 1 (de aquí la imposición anterior de que $|c_1|^2 + |c_2|^2 = 1$).

[1] El producto escalar usual de $\mathbb{C}^2$ es $\langle (z_1, z_2), (z_1', z_2') \rangle = z_1 \overline{z_1'} + z_2 \overline{z_2'}$, donde $\bar{\cdot}$ denota la conjugación compleja.

[2] Sobre $\mathbb{C}^2$, se define la norma usual a partir del producto escalar usual: $\|(z_1, z_2)\| = \sqrt{\langle (z_1, z_2), (z_2, z_2) \rangle} = \sqrt{z_1 \overline{z_1} + z_2 \overline{z_2}} = \sqrt{|z_1|^2 + |z_2|^2}$.

Varios qubits se modelan matemáticamente usando el producto tensorial[1]. $(\mathbb{C}^2)^{\otimes n}$, 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, ver A). 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, escribiremos $|i_1\rangle\,|i_2\rangle = |i_1 i_2\rangle$, $i_1, i_2 \in \{0, 1\}$. También denotaremos $|x\rangle$, $x \in \{0, \ldots, 2^n - 1\}$ a un estado básico de n qubits, entendiendo el número natural x como su representación binaria de n dígitos (por tanto, producto tensorial de n qubits).

La observación de varios qubits es análoga a la situación anterior. Por ejemplo, para 2 qubits, tendríamos que hay una probabilidad $|c_{00}|^2$ de medir un 0 en el primer qubit y un 0 en el segundo, una probabilidad $|c_{10}|^2$ 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 2.2. *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 $|x_{k+1}\rangle = \beta_0\,|0\rangle + \beta_1\,|1\rangle$,

[1] Más detalles pueden consultarse en el apéndice A de este trabajo.

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

Por tanto sacando factor común,

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

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,

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

□

2.2.2 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 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

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

¿Qué aplicaciones lineales verifican esto? Sobre $\mathbb{C}$, la respuesta está en las aplicaciones *unitarias*.

Definición 2.2. Sea X un espacio de Hilbert complejo de dimensión n y sea $f : X \to X$ una aplicación lineal. Se dice que A es *unitaria* si su matriz asociada A verifica que

$$A^\dagger A = I_n$$

Donde I_n es la matriz identidad de orden n, y $A^\dagger$ denota la matriz traspuesta conjugada de A.

Teorema 2.2. *Sea X un espacio de Hilbert complejo de dimensión n y sea $f : X \to X$ una aplicación lineal. Se tiene que f es unitaria si y sólo si para todo $x \in X$,*

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

La demostración de este resultado es una consecuencia del estudio de los operadores en espacios de Hilbert en Análisis Funcional, que puede consultarse en [9].

La definición de una aplicación lineal unitaria con matriz asociada A, 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 2.2. 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 2.3. 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:

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

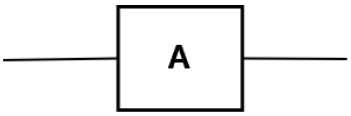
$$H\left|1\right\rangle = \frac{1}{\sqrt{2}}\left|0\right\rangle - \frac{1}{\sqrt{2}}\left|1\right\rangle$$

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

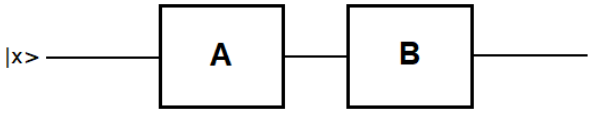
$$H\left(\frac{1}{\sqrt{2}}\left|0\right\rangle - \frac{1}{\sqrt{2}}\left|1\right\rangle\right) = \left|1\right\rangle$$

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:

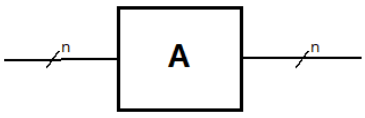


Normalmente, por convenio se supone que la computación ocurre de izquierda a derecha. De esta manera, el siguiente circuito indica la operación de aplicar A y luego B sobre un qubit $|x\rangle$, es decir, $B(A(x))$:



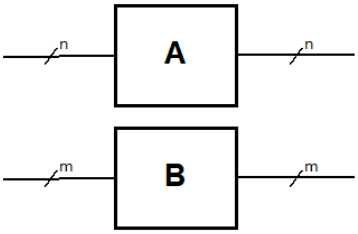
Estas dos operaciones se dice que son *secuenciales*, porque ocurre una después de la otra. Notemos que podríamos haber simplificado este circuito calculando el operador $C = B \circ A$, cuya matriz asociada viene dada de multiplicar BA, las matrices asociadas de B y de A, y reescribiendo el circuito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

Cuando tengamos un operador que 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 $C = A \otimes B$, cuya entrada son $n+m$ qubits, de manera que la acción del circuito podría escribirse como $C(|x\rangle \otimes |y\rangle) = (A \otimes B)(|x\rangle \otimes |y\rangle)$, donde la operación matricial $A \otimes B$ representa el producto de Kronecker de las matrices A y B[1].

Ejemplo 2.4. 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:

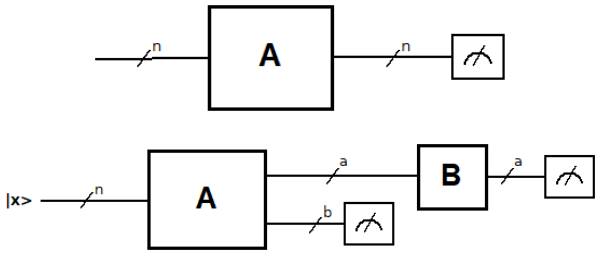
$H|x_1\rangle \otimes H|x_2\rangle \otimes \cdots \otimes H|x_n\rangle = H|0\rangle \otimes H|0\rangle \otimes \cdots \otimes H|0\rangle = \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right) \otimes \cdots \otimes \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right)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

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 [10].

Una puerta especial es la de *medición*. Cuando se coloca al final de uno o varios qubits, pasan de ser qubits 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:

[1] Para más detalles, consultar el apéndice A.



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 [7]).

Ejemplo 2.5. 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.

2.2.3 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 un esquema del algoritmo, supondremos que el orden del elemento buscado, r, es una potencia de 2. El caso general requiere usar un algoritmo de fracciones continuas (ver [7]) 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 $0 \leq j \leq N-1$

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

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 2.3. *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:

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

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

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

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

  Puesto que $k - l \in \mathbb{Z}$, $\left(e^{\frac{2\pi ij}{N}(k-l)} \right)^N = e^{2\pi ij(k-l)} = 1$ 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 ([7]). 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. 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.

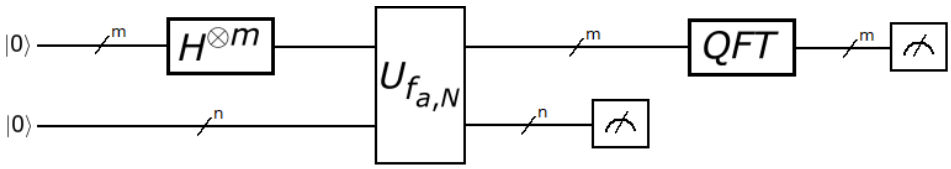


Figura 2.1: El circuito para encontrar el periodo de la función $f_{a,N}$.

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

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

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

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

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 2.5, el estado del sistema será

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

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 $k = 0, \cdots, \frac{2^m}{r}-1$:

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

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 $A = \frac{1}{\sqrt{\frac{2^m}{r}}}$.

$$\begin{aligned} QFT\left(\frac{\sum_{k=0}^{\frac{2^m}{r}-1} |t_0 + kr\rangle}{\sqrt{\frac{2^m}{r}}}\right) &= A \sum_{k=0}^{\frac{2^m}{r}-1} \sum_{j=0}^{2^m-1} e^{\frac{2\pi i(t_0+kr)j}{2^m}} |j\rangle = \\ = A \sum_{j=0}^{2^m-1} \sum_{k=0}^{\frac{2^m}{r}-1} e^{\frac{2\pi i(t_0+kr)j}{2^m}} |j\rangle &= A \sum_{j=0}^{2^m-1} e^{\frac{2\pi i t_0 j}{2^m}} \left(\sum_{k=0}^{\frac{2^m}{r}-1} e^{\frac{2\pi i k r j}{2^m}}\right) |j\rangle \end{aligned} \tag{2.2}$$

Veamos cuánto vale el sumatorio entre paréntesis de la expresión (2.2).

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

$$\sum_{k=0}^{\frac{2^m}{r}-1} e^{\frac{2\pi i k r j}{2^m}} = \sum_{k=0}^{\frac{2^m}{r}-1} e^{\frac{2\pi i k r \lambda \frac{2^m}{r}}{2^m}} = \sum_{k=0}^{\frac{2^m}{r}-1} e^{2\pi i k\lambda} = \sum_{k=0}^{\frac{2^m}{r}-1} 1 = \frac{2^m}{r}$$

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

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

Por tanto, podemos reescribir (2.2) como

$$A \sum_{\lambda=0}^{\lfloor r/2^m - 1 \rfloor} \frac{2^m}{r} e^{\frac{2\pi i t_0 \lambda \frac{2^m}{r}}{2^m}} \left|\lambda\frac{2^m}{r}\right\rangle = A \sum_{\lambda=0}^{\lfloor r/2^m - 1 \rfloor} \frac{2^m}{r} e^{\frac{2\pi i t_0 \lambda}{r}} \left|\lambda\frac{2^m}{r}\right\rangle \tag{2.3}$$

Como podemos ver, la acción del operador QFT es eliminar el término inicial t_0 y cambiar la longitud del periodo de r a $\frac{2^m}{r}$.

6. Medimos los m qubits. A partir de la ecuación (2.3) 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. A partir de varias medidas, obtendremos resultados para distintos valores de k, a través de los cuales podemos hallar valor r buscado.

2.2.4 Ejemplo para $N = 15$

Finalmente, veremos un ejemplo práctico para factorizar $N = 15$. Ya que los desarrollos del estado del sistema son largos, los resultados aquí presentados han sido obtenidos con ayuda informática (ver apéndice B para el código en *Mathematica* usado). Con la notación de la sección anterior, elegimos, por ejemplo, $a = 7$; ya que $\mathrm{mcd}(7, 15) = 1$ podemos continuar. En este caso, $n = \lceil \log_2 N \rceil = 4$, y $m = 2n = 8$.

1. El estado inicial del sistema es $|0, 0\rangle$.

2. Tras aplicar la puerta de Hadamard, el estado del ordenador es

$$\frac{1}{\sqrt{2^8}}\left(|0,0\rangle + |1,0\rangle + |2,0\rangle + \cdots + |255,0\rangle\right) = \frac{1}{16}\left(|0,0\rangle + |1,0\rangle + |2,0\rangle + \cdots + |255,0\rangle\right)$$

3. Tras aplicar el operador $U_{f_{7,15}}$, el estado queda:

$$\begin{aligned}\frac{1}{16}(&|0,1\rangle + |1,7\rangle + |2,4\rangle + |3,13\rangle + |4,1\rangle + |5,7\rangle \\ &+ |6,4\rangle + |7,13\rangle + |8,1\rangle + \cdots + |255,13\rangle)\end{aligned}$$

4. Al medir los últimos 4 qubits, podemos medir $1, 7, 4$ ó 13 con igual probabilidad: como en el estado del ordenador cada término aparece con la misma frecuencia, es decir, exactamente $\frac{256}{4} = 64$ veces, la probabilidad de medir cada uno será $64 \cdot \left|\frac{1}{16}\right|^2 = \frac{1}{4}$. Supongamos que medimos 13. Entonces, el estado del ordenador tras realizar esta medición, será:

$$\frac{1}{\sqrt{64}} \left(|3, 13\rangle + |7, 13\rangle + |11, 13\rangle + |15, 13\rangle + \cdots + |255, 13\rangle\right)$$

Ya que en total hay $\frac{256}{4} = 64$ términos, todos con amplitud 1; luego la norma del qubit sin normalizar sería $\sqrt{64 \cdot |1|^2} = 8$.

5. Aplicamos la transformada cuántica de Fourier a los restantes 8 qubits no medidos,en estado:

$$\frac{1}{8} \left(|3\rangle + |7\rangle + |11\rangle + |15\rangle + \cdots + |255\rangle\right)$$

Así, obtenemos el estado:

$$\frac{1}{2} |0\rangle - \frac{i}{2} |64\rangle - \frac{1}{2} |128\rangle + \frac{i}{2} |192\rangle$$

6. Finalmente, medimos los qubits restantes del sistema. En este caso, podremos medir 0, 64, 128 ó 192. La probabilidad de medir cada uno de estos estados es $\frac{1}{4}$ (por ejemplo, para $|64\rangle$, $\left|\frac{-i}{2}\right|^2 = \frac{1}{4}$). Supongamos que medimos $x = 128$, entonces, calculamos

$$\frac{x}{2^m} = \frac{128}{256} = \frac{1}{2}$$

De aquí, el orden del elemento debe ser un múltiplo de 2. Pero $7^2 \equiv 4 \mod 15$. Si repetimos el algoritmo para realizar otra medida y medimos, por ejemplo, $x = 196$, entonces

$$\frac{x}{2^m} = \frac{192}{256} = \frac{3}{4}$$

De aquí, el orden del elemento debe ser múltiplo de 4. Y ya que $7^4 \equiv 1 \mod 15$, es claro que el orden buscado es $r = 4$.

El orden que devuelve el ordenador cuántico es par. Además, $7^{4/2} \equiv 7^2 \equiv 4 \not\equiv 14 \mod 15$, por tanto, podemos hallar los factores de $N = 15$. En este caso,

$$\mathrm{mcd}(7^2 + 1 \mod 15, 15) = 5$$

$$\mathrm{mcd}(7^2 - 1 \mod 15, 15) = 3$$

Lo que nos proporciona los dos factores de 15, y el algoritmo termina.

2.3 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.

2.3.1 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 2.4. *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 2.3). 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 $(-1)^{f(x)}\,|x\rangle$, siempre y cuando $|q\rangle = H(|1\rangle)$.

2.3.2 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 ($|0\rangle, \ldots, |N-1\rangle$). Si así no fuera, podemos "completar" el conjunto hasta la siguiente potencia de 2 repitiendo algunos de sus elementos o añadiendo elementos que no son soluciones. El esquema del circuito cuántico para el algoritmo de Grover se resume en la siguiente figura.

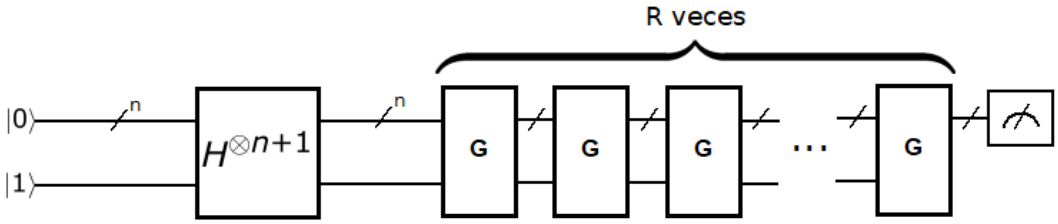


Figura 2.2: El esquema general del circuito para ejecutar el algoritmo de Grover.

Como podemos observar, comenzamos con n qubits en el estado básico $|0\rangle$. El último qubit se pone en el estado básico $|1\rangle$. De esta manera, el estado del ordenador tras aplicar $H^{\otimes n+1}$ es $H^{\otimes n}\,|0\rangle \otimes H\,|1\rangle$, lo cual nos permite aplicar la Proposición 2.4 para simplificar

la acción del oráculo en el resto del circuito. 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 N y M). Su estructura es la siguiente:

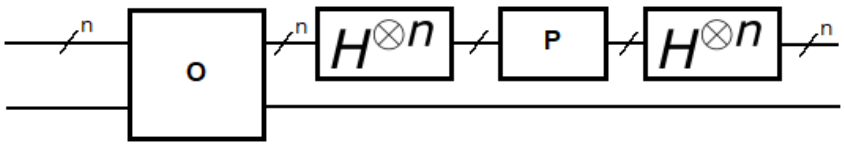


Figura 2.3: El esquema del operador de Grover.

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 2.5. *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:

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

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

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

2.3.3 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 [7]. 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

$$|\psi\rangle = \sqrt{\frac{N-M}{N}} |\alpha\rangle + \sqrt{\frac{M}{N}} |\beta\rangle = a|\alpha\rangle + b|\beta\rangle \tag{2.4}$$

A partir de la Proposición 2.4 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 $|\beta\rangle$ [7]. 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 [7], 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 (2.4), 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:

$$R = CI\left(\frac{\arccos\sqrt{\frac{M}{N}}}{\theta}\right) \tag{2.5}$$

La expresión (2.5) 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 $M < N/2$. 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

3

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 eficientes conocidos 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*, y del criptosistema *ElGamal*).

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 decodificar una palabra de un código autocorrector (criptografía basada en códigos).

3.1 Criptografía basada en funciones hash

3.1.1 Introducción a las funciones hash

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

Definición 3.1. 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 3.2. 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 3.3. Una *función hash sin clave* es una función $h : \mathcal{X} \to \mathcal{Y}$, con $\mathcal{X}$ y $\mathcal{Y}$ definidas en la Definición 3.1.

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 3.4. 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 3.5. 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 3.6. 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 3.5 y la Definición 3.6: mientras que en la primera tenemos un valor x fijado de antemano, en la segunda no. De hecho, la resistencia a la colisión implica la resistencia a la segunda preimagen, pero el recíproco no es cierto en general [11].

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*) [12]. Para otras versiones más antiguas, sin embargo, como el *SHA-1*, se han encontrado ataques que producen colisiones [13]. En la siguiente sección, presentamos un ejemplo de función hash.

3.1.2 Un ejemplo de función hash: SHA-1

El ejemplo que vamos a ver es el de la función *SHA-1*, tomado de [14], pág. 129-139; como hemos indicado anteriormente, esta función ya no está en uso, pero tiene una descripción más simple que las citadas *SHA-256*, *SHA-384* y *SHA-512*. Al igual que estas últimas, se define a través de un proceso, denominado la *construcción de Merkle-Damgård*, a partir de

una función de compresión[1].

La función *SHA-1* está definida sobre sucesiones de a lo más $2^{64} - 1$ bits (aunque teóricamente una función hash está definida sobre sucesiones de bits de cualquier longitud, al menos en el momento en el que estuvo en uso la función *SHA-1*, $2^{64} - 1$ era una cota considerablemente alta en la práctica).

Presentamos primero la función de compresión de *SHA-1*, que denotamos por

$$\mathbf{f} : \{0,1\}^{160+512} \to \{0,1\}^{160}$$

Para ello, haremos uso de las siguientes operaciones básicas en $\{0,1\}$:

- $X \wedge Y$ definida como: $0 \wedge 0 = 0$, $0 \wedge 1 = 0$, $1 \wedge 0 = 0$, $1 \wedge 1 = 1$,
- $X \vee Y$ definida como: $0 \vee 0 = 0$, $0 \vee 1 = 1$, $1 \vee 0 = 1$, $1 \vee 1 = 1$,
- $X \oplus Y$ definida como: $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, $1 \oplus 1 = 0$,
- $\neg X$ definida como: $\neg 0 = 1$, $\neg 1 = 0$,

y que extendemos a $\{0,1\}^{32}$ realizando las operaciones bit a bit. Definimos además la operación en $\{0,1\}^{32}$:

- $X + Y$,

como sigue: a X e Y se asocian los números naturales a, b cuyos desarrollos binarios son X, Y respectivamente; se define $X + Y$ como el desarrollo binario de $a + b \bmod 2^{32}$. Por último, se considera la función de $\{0,1\}^{32}$ en $\{0,1\}^{32}$

- $\mathbf{ROTL}^s(X)$,

para $0 \leqslant s \leqslant 31$, que es la traslación circular a la izquierda de s posiciones en X.

Sean $P \in \{0,1\}^{160}$, $Q \in \{0,1\}^{512}$. Escribimos

$$P = H_0 \parallel H_1 \parallel H_2 \parallel H_3 \parallel H_4\,,$$

[1]Una propiedad importante de la construcción de Merkle-Damgård es que conserva la resistencia a la colisión, es decir, si usando la construcción de Merkle-Damgård, se obtiene una función hash a partir de una función de compresión resistente a la colisión, entonces la función hash obtenida también es resistente a la colisión.

donde H_i son sucesiones de 32 bits, y $X \,\|\, Y$ denota la concatenación de las sucesiones X, Y. Análogamente,

$$Q = W_0 \,\|\, W_1 \,\|\, \ldots \,\|\, W_{15}\,,$$

donde W_i son sucesiones de 32 bits. Se definen las funciones $\mathbf{f}_t$, $0 \leqslant t \leqslant 79$, de $\{0,1\}^{32} \times \{0,1\}^{32} \times \{0,1\}^{32}$ en $\{0,1\}^{32}$ como:

$$\mathbf{f}_t(B,C,D) = \begin{cases} (B \wedge C) \vee (\neg B \wedge D) & \text{si } 0 \leqslant t \leqslant 19\,, \\ B \oplus C \oplus D & \text{si } 20 \leqslant t \leqslant 39\,, \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & \text{si } 40 \leqslant t \leqslant 59\,, \\ B \oplus C \oplus D & \text{si } 60 \leqslant t \leqslant 79\,. \end{cases}$$

Se definen las sucesiones de 32 bits K_t (escritas en hexadecimal), $0 \leqslant t \leqslant 79$, como

$$K_t = \begin{cases} \texttt{5A827999} & \text{si } 0 \leqslant t \leqslant 19\,, \\ \texttt{6ED9EBA1} & \text{si } 20 \leqslant t \leqslant 39\,, \\ \texttt{8F1BBCDC} & \text{si } 40 \leqslant t \leqslant 59\,, \\ \texttt{CA62C1D6} & \text{si } 60 \leqslant t \leqslant 79\,. \end{cases}$$

Para $16 \leqslant t \leqslant 79$, se define W_t por recurrencia como

$$W_t = \mathbf{ROTL}^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})\,.$$

Partiendo de $P = H_0 \,\|\, H_1 \,\|\, H_2 \,\|\, H_3 \,\|\, H_4$ y $Q = W_0 \,\|\, W_1 \,\|\, \ldots \,\|\, W_{15}$, aplicamos el siguiente algoritmo:

Algoritmo 3.1. Función de compresión f

Entrada: $P \in \{0,1\}^{160}, Q \in \{0,1\}^{512}$.
Salida: Un elemento de $\{0,1\}^{160}$.

1. Poner $A := H_0$.
2. Poner $B := H_1$.
3. Poner $C := H_2$.
4. Poner $D := H_3$.
5. Poner $E := H_4$.
6. **Para cada** $t = 0$ **hasta** 79 **hacer:**
   (a) Calcular $temp := \mathbf{ROTL}^5(A) + \mathbf{f}_t(B,C,D) + E + W_t + K_t$.
   (b) Poner $E := D$.
   (c) Poner $D := C$.

(d) Calcular $C := \mathbf{ROTL}^{30}(B)$.
(e) Poner $B := A$.
(f) Poner $A := temp$.
7. Poner $H_0 := H_0 + A$.
8. Poner $H_1 := H_1 + B$.
9. Poner $H_2 := H_2 + C$.
10. Poner $H_3 := H_3 + D$.
11. Poner $H_4 := H_4 + E$.
12. **Devolver** $H_0 \parallel H_1 \parallel H_2 \parallel H_3 \parallel H_4$.

Vemos ahora la construcción de Merkle-Damgård. Partimos de la función de compresión $\mathbf{f} : \{0,1\}^{160+512} \to \{0,1\}^{160}$ definida antes, y vamos a definir la función *SHA-1* : $\mathcal{X} \to \{0,1\}^{160}$, donde

$$\mathcal{X} = \bigcup_{i=0}^{2^{64}-1} \{0,1\}^i .$$

Sea $x \in \mathcal{X}$; entonces $x \in \{0,1\}^i$ para cierto $0 \leq i \leq 2^{64} - 1$. Sea $d \equiv 447 - i \mod 512$, $d \geqslant 0$, y sea l la representación binaria de i de longitud 64, es decir, $l \in \{0,1\}^{64}$. Definimos el "acolchado" de x como:

$$PAD(x) = x \parallel 1 \parallel 0^d \parallel l .$$

$PAD(x)$ tiene un número de bits múltiplo de 512 ya que la longitud de $PAD(x)$ es $i + 1 + d + 64 \equiv i + 1 + 447 - i + 64 \equiv 512 \equiv 0 \mod 512$; luego podemos escribir

$$PAD(x) = y_1 \parallel y_2 \parallel \ \ldots \ \parallel y_n ,$$

donde y_i es una sucesión de 512 bits. Consideramos ahora las sucesiones iniciales de 32 bits (escritas en hexadecimal) siguientes:

$$\begin{aligned} H_0 &= \texttt{67452301} , \\ H_1 &= \texttt{EFCDAB89} , \\ H_2 &= \texttt{98BADCFE} , \\ H_3 &= \texttt{10325476} , \\ H_4 &= \texttt{C3D2E1F0} . \end{aligned}$$

Sea $z_0 = H_0 \parallel H_1 \parallel H_2 \parallel H_3 \parallel H_4$. Para $i = 1, \ldots, n$, definimos por recurrencia[1]

$$z_i = \mathbf{f}(z_{i-1}, y_i) .$$

[1] La construcción de Merkle-Damgård sobre la función de compresión $\mathbf{f}$ es la recurrencia $z_i = \mathbf{f}(z_{i-1}, y_i)$, partiendo de la sucesión inicial z_0.

Finalmente *SHA-1*$(x) = z_n$.

La familia de funciones hash *SHA-3* es una alternativa a la familia *SHA-2* (formada por las citadas *SHA-256*, *SHA-384* y *SHA-512*, que aún siguen en uso). Una de las cosas novedosas de la familia *SHA-3* es que no utiliza la construcción de Merkle-Damgård.

3.1.3 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. En lo que sigue, denotaremos

$$\{0,1\}^* = \bigcup_{i=0}^{\infty} \{0,1\}^i$$

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 $j = 0, \ldots, n-1$:

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

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')

[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.

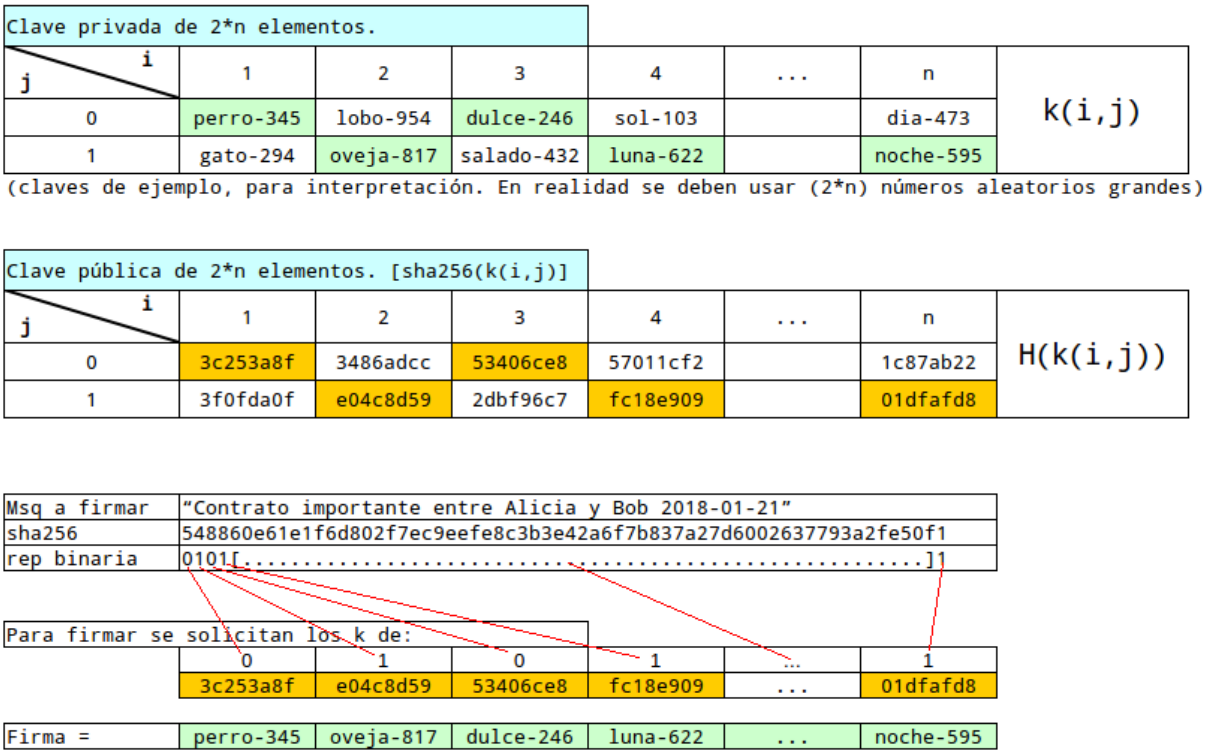


Figura 3.1: Esquema del proceso de firma en el esquema LD-OTS. La verificación se hace calculando la función hash del mensaje firmado, y comprobando que el hash de cada elemento de la firma coincide con la clave pública en cada bit del hash del mensaje firmado. Imagen de Ignacio Zelaya, bajo licencia Creative Commons Attribution-Share Alike 4.0 International.

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.

3.1.4 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 3.2. 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 3.3. 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*.

3.1.5 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 [11].

Definición 3.7. 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 3.8. Un *adversario*, denotado ADV, es cualquier algoritmo probabilístico con cualquier número de entradas.

Por ejemplo, los Algoritmos 3.2 y 3.3 presentados en la sección anterior son ejemplos de adversarios, que podríamos haber denotado ADV_{Bruta} y $\text{ADV}_{2documentos}$, por ejemplo.

Definición 3.9. 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

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

Es decir, que su probabilidad de éxito es, a lo sumo, ε.

De manera similar, podrían extenderse las Definiciones 3.5 y 3.6.

Definición 3.10. 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* $\mathcal{O}(s_k, \cdot) : \mathcal{P} \to \mathcal{A}$ como una función que dado un documento $x \in \mathcal{P}$, devuelve su firma $\sigma \in \mathcal{A}$.

Teóricamente, un oráculo de firma hace la misma función que el algoritmo de firma del sistema. Sin embargo, a diferencia de éste, un atacante tiene acceso al oráculo, aun sin conocer la clave privada, y 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 3.11. 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 $\mathcal{O}(s_k, \cdot)$ un oráculo de firma del sistema. Definimos un *falsificador* $\textsc{For}^{\mathcal{O}(s_k,\cdot)}(p_k)$ 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

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

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 documento con cada par de claves. Finalmente, definimos que un sistema de firma sea *existencialmente irrompible por un ataque de mensajes elegidos adaptativamente*.

Definición 3.12. 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 que haga como máximo q consultas al oráculo de firma 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. En la demostración del teorema, identificaremos los mensajes a firmar como su imagen por la función hash, $H(M)$, con tal de simplificar la notación. De esta manera, todo documento a firmar $M \in \{0,1\}^n$ (el espacio de salida de H, tal y como se especifica en el esquema de la sección 3.1.3).

Teorema 3.1. *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 3.4. 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 privada y pública (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 pública 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_{ADV} y la probabilidad de éxito ε_{ADV} 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 el tiempo que tarda en ejecutarse este adversario es

$$t_{ADV} = t + t_{GEN} + t_{SIG} \tag{3.1}$$

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

$$\varepsilon_{ADV} \geq \frac{1}{2} \cdot \varepsilon \cdot \frac{1}{2n} = \frac{\varepsilon}{4n} \tag{3.2}$$

Sin embargo, ya que por hipótesis $t = t_{OW} - t_{SIG} - t_{GEN}$, de la ecuación (3.1) deducimos que $t_{ADV} = t_{OW}$, y como $\varepsilon > 4n \cdot \varepsilon_{OW}$, de la ecuación (3.2) deducimos que:

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

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*.

3.1.6 El sistema de firma basado en árboles hash de Merkle

El sistema de firma de Lamport-Diffie en la práctica no es útil, puesto que sólo es posible firmar un solo documento con cada par de claves. Sin embargo, Ralph Merkle propuso en 1979 una solución a este problema [15]. La idea es extender cualquier sistema de firma de un solo uso, generando un número arbitrario pero finito de pares de claves y reducir la validez de cada una de ellas a una sola clave pública. De esta manera, cuando un usuario

Alice quiera firmar un documento, incluye la firma y la clave pública con la que fue firmada. Un usuario, Bob, solo debe comprobar que la firma es correcta mediante el algoritmo de verificación del sistema de un solo uso, usando la clave pública que proporcionó Alice junto con la firma; y seguidamente comprobar que la clave pública es auténtica usando la clave pública general de Alice.

El esquema de firma de Merkle (MSS) funciona con cualquier firma de un solo uso (por ejemplo, la firma de Lamport-Diffie) y con una función hash, que denotaremos

$$g : \{0,1\}^* \to \{0,1\}^n$$

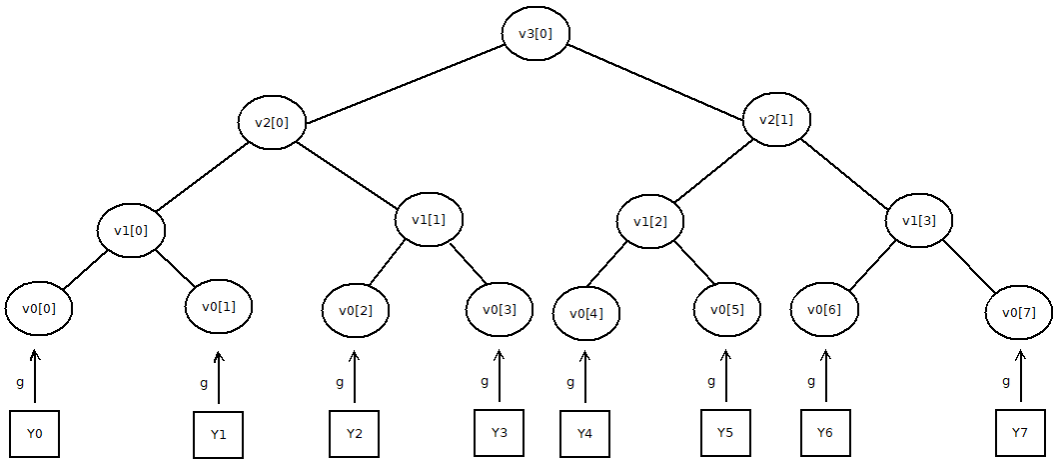
Estudiaremos cada parte del esquema por separado, dada su extensión.

3.1.6.1 Generación de claves

Primero se selecciona un número natural $H \geq 2$. La cantidad de documentos que se pueden firmar con este sistema son 2^H. Seguidamente, se generan 2^H pares de claves privada y pública del sistema de firma de un solo uso (X_j, Y_j), $0 \leq j < 2^H$. A continuación, se construye una estructura de datos denominada "árbol de Merkle". En la base del árbol, se encuentran las imágenes $g(Y_j)$ de las claves públicas. Luego, se prosigue construyendo los demás niveles hasta llegar a la raíz del árbol; en cada nivel se construye un nodo a partir de los dos inmediatamente inferiores a partir de la siguiente expresión:

$$v_h[j] = g(v_{h-1}[2j]||v_{h-1}[2j+1]), \qquad 1 \leq h \leq H, 0 \leq j < 2^{H-h}$$

donde h es el nivel del árbol, y j el número de nodo de ese nivel y $||$ denota la operación de concatenación. Por ejemplo, así sería la estructura de un árbol de Merkle con $H = 3$.



La clave pública del sistema es la raíz del árbol, $v_H[0]$, y la clave privada es la estructura del árbol completa, con los respectivos pares de claves privadas y públicas del sistema de un solo uso subyacente.

3.1.6.2 Algoritmo de firma

Sea M un documento a firmar. Supongamos que ya han sido utilizadas s firmas anteriormente, $0 \leq s \leq 2^H - 1$. Por tanto, este documento se firmará con el par de claves (X_s, Y_s). Se firma el documento usando el esquema de un solo uso, obteniendo así la firma σ_{OTS}. Seguidamente, se calcula una *ruta de autentificación*, que es una secuencia $A_s = (a_0, \ldots, a_{H-1})$ de $H - 1$ nodos del árbol de Merkle. Cada a_h, $0 \leq h \leq H - 1$ es el "hermano" de cada nodo en el camino del árbol desde el nodo base usado a la raíz del árbol:

$$a_h = \begin{cases} v_h[\lfloor s/2^h \rfloor - 1] & \text{si} \quad \lfloor s/2^h \rfloor \text{ es impar} \\ v_h[\lfloor s/2^h \rfloor + 1] & \text{si} \quad \lfloor s/2^h \rfloor \text{ es par} \end{cases}$$

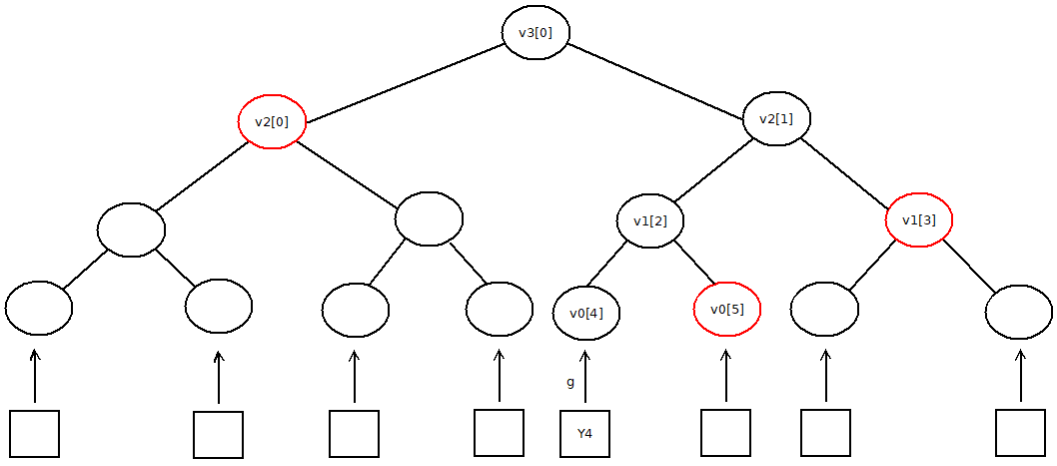


Figura 3.2: Ruta de autentificación para un árbol de Merkle con $H = 3$, $s = 4$. Los nodos señalados en rojo son los nodos que se incluyen en la ruta.

La firma del documento, finalmente, es

$$\sigma = (\sigma_{OTS}, s, Y_s, A_s)$$

3.1.6.3 **Algoritmo de verificación**

Para verificar un documento firmado (M, σ), primero se comprueba que la firma σ_{OTS} es correcta usando el algoritmo de verificación del sistema de un solo uso y la clave pública Y_s proporcionada en σ. A continuación, se verifica que Y_s es correcta. Para ello, reconstruye el camino hasta la raíz del árbol usando la ruta de autentificación A_s: se calculan unos elementos p_h, $0 \leq h \leq H$ de la siguiente manera:

$$p_h = \begin{cases} g(a_{h-1}||p_{h-1}) & \text{si} \quad \lfloor s/2^{h-1} \rfloor \text{ es impar} \\ g(p_{h-1}||a_{h-1}) & \text{si} \quad \lfloor s/2^{h-1} \rfloor \text{ es par} \end{cases}$$

para $1 \leq h \leq H$, y $p_0 = g(Y_s)$. La firma es correcta si p_H es igual a la clave pública del sistema.

La seguridad de este esquema radica en que un atacante que quisiese falsificar un documento, suponiendo que el sistema de un solo uso es seguro, generaría un par de claves (X', Y') y tendría que calcular una ruta de autentificación A'. Sin embargo, por la propiedad de resistencia a la preimagen de la función hash g, no puede revertir el proceso para encontrar un nodo a_{H-1} tal que $g(a_{H-1}||p_{H-1}) = v_H[0]$, es decir, que coincida con la clave pública del sistema, y de igual manera con el resto de niveles del árbol. Solo Alice, que conoce la estructura del árbol completa, puede generar estas rutas de autentificación.

3.2 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 un tipo de códigos autocorrectores especiales, los *códigos Goppa*.

3.2.1 Introducción a los códigos Goppa

Antes de presentar los códigos Goppa y sus propiedades, recordaremos algunas definciones básicas sobre códigos.

Definición 3.13. 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 3.14. 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 $\mathcal{C}^\perp$.

Observemos que un $[n, k]$-código lineal $\mathcal{C}$ puede definirse a través de una matriz de control de paridad H como $\mathcal{C} = \{x \in \mathbb{F}_q^n : Hx = 0\}$, lo cual define al código según unas ecuaciones implícitas sobre $\mathbb{F}_q^n$. Por extensión, llamaremos también matriz de control de paridad a la matriz de coeficientes de cualquier sistema de ecuaciones que defina a $\mathcal{C}$ (no necesariamente sobre $\mathbb{F}_q$, ni de orden $(n-k)\times n$).

Definición 3.15. 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 3.16. 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 defenición de un código Goppa. Aunque existen diversas definiciones, nosotros seguiremos [16].

Definición 3.17. Sea $g(z)$ un polinomio mónico de grado t sobre $\mathbb{F}_{q^m}$. Sea $L = \{\gamma_0, \gamma_1, \ldots, \gamma_{n-1}\} \subset \mathbb{F}_{q^m}$, $|L| = n$, 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

$$\sum_{i=0}^{n-1} \frac{c_i}{z - \gamma_i} \equiv 0 \mod g(z) \tag{3.3}$$

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 (3.3). Así, podemos dar el siguiente resultado (las demostraciones están tomadas de [16]):

Proposición 3.1. *Tomando* $h_j := \frac{1}{g(\gamma_j)}$, $j = 0, \ldots, n-1$, *una matriz de control de paridad de* $\Gamma(L, g)$ *es:*

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

Demostración. Vamos a comprobar que $\Gamma(L, g)$ es el conjunto de palabras $(c_0, c_1, \ldots, c_{n-1}) \in \mathbb{F}_q^n$ tales que

$$\begin{pmatrix} h_0 & h_1 & \cdots & h_{n-1} \\ h_0\gamma_0 & h_1\gamma_1 & \cdots & h_{n-1}\gamma_{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ h_0\gamma_0^{t-1} & h_1\gamma_1^{t-1} & \cdots & h_{n-1}\gamma_{n-1}^{t-1} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Como $g(\gamma_j) \neq 0$, $\mathrm{mcd}(z - \gamma_j, g(z)) = 1$. Del lema de Bézout se deduce que la clase de $z - \gamma_j$ en el anillo cociente $\mathbb{F}_{q^m}[z]/\langle g(z)\rangle$ es una unidad[1]; un representante de su inverso es[2]

$$-h_j \cdot \frac{g(z) - g(\gamma_j)}{z - \gamma_j},$$

donde $h_j = 1/g(\gamma_j)$. Sea $g(z) = g_0 + g_1 z + \ldots + g_t z^t$ (donde $g_t = 1$); entonces

$$\frac{g(z) - g(\gamma_j)}{z - \gamma_j} = z^{t-1} + \sum_{i=1}^{t-1} \left(\gamma_j^i + \gamma_j^{i-1} g_{t-1} + \ldots + g_{t-i}\right) z^{t-i-1}.$$

[1] Ya que existen dos polinomios $a(z), b(z) \in \mathbb{F}_{q^m}[z]/\langle g(z)\rangle$ tales que $a(z)(z - \gamma_j) + b(z)g(z) = 1$; tomando clases concluimos que $a(z)(z - \gamma_j) = 1$, lo que significa que es una unidad.

[2] Basta multiplicar y tomar clases para ver que el producto con $(z - \gamma_j)$ vale 1.

La condición $\sum_{j=0}^{n-1} \frac{c_j}{z-\gamma_j} \equiv 0 \text{ mód } g(z)$ que define $\Gamma(L,g)$ equivale a

$$\sum_{j=0}^{n-1} -c_j h_j \cdot \frac{g(z)-g(\gamma_j)}{z-\gamma_j} = 0\,, \tag{3.4}$$

pues el lado izquierdo de la ecuación (3.4) es un polinomio en z de grado menor que t. Esta ecuación es equivalente al sistema, igualando los coeficientes del miembro izquierdo de (3.4) a 0:

$$H \begin{pmatrix} c_0 \\ \vdots \\ c_{n-1} \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix},$$

donde

$$H = \begin{pmatrix} h_0 & \cdots & h_{n-1} \\ h_0(g_{t-1}+\gamma_0) & \cdots & h_{n-1}(g_{t-1}+\gamma_{n-1}) \\ \vdots & \ddots & \vdots \\ h_0(g_1+g_2\gamma_0+\ldots+\gamma_0^{t-1}) & \ldots & h_{n-1}(g_1+g_2\gamma_{n-1}+\ldots+\gamma_{n-1}^{t-1}) \end{pmatrix}.$$

Aplicando Gauss a H (simplificamos primero la segunda fila y después hacia abajo hasta simplificar la última fila), concluimos que la ecuación (3.4) es equivalente a

$$\begin{pmatrix} h_0 & \cdots & h_{n-1} \\ h_0\gamma_0 & \cdots & h_{n-1}\gamma_{n-1} \\ \vdots & \ddots & \vdots \\ h_0\gamma_0^{t-1} & \ldots & h_{n-1}\gamma_{n-1}^{t-1} \end{pmatrix} \begin{pmatrix} c_0 \\ \vdots \\ c_{n-1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

que es lo que queríamos justificar. □

Sin exigir restricciones al código, tememos las siguientes cotas para la dimensión y distancia mínima de un código de Goppa:

Proposición 3.2. *Con la notación anterior, un código de Goppa $\Gamma(L,g)$ tiene dimensión $k \geq n - mt$ y distancia mínima $d \geq t+1$.*

Demostración. Consideramos la matriz

$$H' = \begin{pmatrix} h_0 & h_1 & \cdots & h_{n-1} \\ h_0\gamma_0 & h_1\gamma_1 & \cdots & h_{n-1}\gamma_{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ h_0\gamma_0^{t-1} & h_1\gamma_1^{t-1} & \cdots & h_{n-1}\gamma_{n-1}^{t-1} \end{pmatrix}.$$

Por la Proposición 3.1, sabemos que $\Gamma(L, g)$ es el conjunto de palabras $(c_0, c_1, \ldots, c_{n-1}) \in \mathbb{F}_q^n$ tales que

$$H' \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}. \tag{3.5}$$

Denotemos $\beta_{ij} = h_j \gamma_j^i \in \mathbb{F}_{q^m}$. Fijamos una base $\{\alpha_1, \ldots, \alpha_m\}$ de $\mathbb{F}_{q^m}$ sobre $\mathbb{F}_q$; escribimos $\beta_{ij} = \sum_{l=1}^{m} b_{ilj}\alpha_l$, $b_{ilj} \in \mathbb{F}_q$. Sea H'' la matriz $tm \times n$ cuyas filas son $(b_{il,0}, \ldots, b_{il,n-1})$. De la ecuación (3.5), deducimos que para todo $i = 1, \ldots, t$, se tiene que

$$\sum_{j=0}^{n-1} \beta_{ij} c_j = 0 \iff \sum_{j=0}^{n-1} \sum_{l=0}^{m-1} \beta_{ilj} \alpha_l c_j = 0 \iff \sum_{l=0}^{m-1} \left(\sum_{j=0}^{n-1} \beta_{ilj} c_j \right) \alpha_l = 0$$

y de esta última expresión es sencillo ver que esto se verifica si y sólo si $\sum_{j=0}^{n-1} \beta_{ilj} c_j = 0$ para todo $i = 1, \ldots, t$, $l = 1, \ldots, m$ (la condición suficiente es trivial, la necesaria se desprende del hecho de que los α_l son linealmente independientes). Por tanto, el código $\Gamma(L, g)$ es el conjunto de palabras $(c_0, c_1, \ldots, c_{n-1}) \in \mathbb{F}_q^n$ tales que

$$H'' \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Luego la dimensión de $\Gamma(L, g)$ es igual a $n - \operatorname{rango}(H'') \geq n - mt$.

Vemos ahora que la distancia mínima de $\Gamma(L, g)$ es mayor o igual que $t+1$. Observamos que cualesquiera t columnas de la matriz H' son linealmente independientes; esto se deduce de que $h_j \neq 0$ para $j = 0, \ldots, n-1$, de que $\gamma_{j_1} \neq \gamma_{j_2}$ si $j_1 \neq j_2$, y de la expresión del determinante de Vandermonde. Deducimos que ninguna palabra $(c_0, c_1, \ldots, c_{n-1}) \in \Gamma(L, g)$ distinta de $(0, \ldots, 0)$ puede tener peso menor que $t + 1$, pues de lo contrario, por la ecuación (3.5), habría una combinación lineal no trivial de t columnas de H' igual a $(0, \ldots, 0)^t$, que contradice lo anterior. Por ser $\Gamma(L, g)$ lineal, concluimos que su distancia mínima es mayor o igual que $t + 1$. □

Sin embargo, exigiendo $q = 2$ y que el polinomio de Goppa g no tenga raíces múltiples (por ejemplo, si es irreducible), llegamos al siguiente resultado.

Proposición 3.3. *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 $d \geq 2t + 1$.*

Demostración. Sea $L = \{\gamma_0, \gamma_1, \ldots, \gamma_{n-1}\} \subset \mathbb{F}_{2^m}$, $|L| = n$. Consideremos el $\mathbb{F}_2$-espacio vectorial

$$\mathcal{R} = \left\{ \sum_{j=0}^{n-1} \frac{b_j}{z - \gamma_j} : (b_0, b_1, \ldots, b_{n-1}) \in \mathbb{F}_2^n \right\} .$$

Sea $\xi(z) \in \mathcal{R}$ función racional no nula; escribimos $\xi(z) = n(z)/d(z)$, donde $n(z), d(z) \in \mathbb{F}_{2^m}[z]$ con $mcd(n(z), d(z)) = 1$. El *peso* de $\xi(z) \in \mathcal{R}$ se define como $wt(\xi(z)) = \deg d(z)$. Dadas dos funciones racionales $\xi(z), \eta(z) \in \mathcal{R}$, $\xi(z) \neq \eta(z)$, se define la *distancia* $d(\xi(z), \eta(z)) = wt(\xi(z) - \eta(z))$. Ahora, la asignación

$$(b_0, b_1, \ldots, b_{n-1}) \mapsto \sum_{j=0}^{n-1} \frac{b_j}{z - \gamma_j}$$

es un isomorfismo de $\mathbb{F}_2^n$ en $\mathcal{R}$ que conserva las distancias[1] (la distancia en $\mathbb{F}_2^n$ es la distancia de Hamming).

Sea $(c_0, c_1, \ldots, c_{n-1}) \in \Gamma(L, g)$ una palabra no nula; consideramos el polinomio $f(z) = \prod_{j=0}^{n-1} (z - \gamma_j)^{c_j}$. Entonces podemos expresar[2]

$$\xi(z) = \sum_{j=0}^{n-1} \frac{c_j}{z - \gamma_j} = f'(z)/f(z) \,,$$

donde $f'(z)$ es la derivada formal de $f(z)$; observamos que $\mathrm{mcd}(f(z), f'(z)) = 1$ pues las raíces de $f(z)$ son simples. Ahora, $\mathrm{mcd}(f(z), g(z)) = 1$ ya que $g(\gamma_j) \neq 0$ para cada j; como $(c_0, c_1, \ldots, c_{n-1}) \in \Gamma(L, g)$, $\xi(z) \equiv 0 \bmod g(z)$, y por tanto $f'(z) \equiv 0 \bmod g(z)$. Como $q = 2$, $f'(z)$ sólo tiene términos de grado par, y por tanto $f'(z) = (u(z))^2$, con $u(z) \in \mathbb{F}_{2^m}[z]$; como $g(z)$ no tiene raíces múltiples, $u(z) \equiv 0 \bmod g(z)$. Concluimos que $\deg f(z) \geq 1 + \deg f'(z) \geq 1 + 2 \deg g(z) = 1 + 2t$; por tanto $wt(\xi(z)) \geq 2t + 1$. □

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 ([17]).

[1] En efecto, ya que los b_j valen 0 ó 1, si $b^{(1)}, b^{(2)}$ son dos palabras a distancia $k > 0$, $d(f(b^{(1)}) - f(b^{(2)})) = d\left(\sum_{j=0}^{n-1} \frac{b_j^{(1)}}{z-\gamma_j} - \frac{b_j^{(2)}}{z-\gamma_j}\right) = d\left(\sum_{j:b_j^{(1)} \neq b_j^{(2)}} \frac{b_j^{(1)} - b_j^{(2)}}{z-\gamma_j}\right)$, y sacando común denominador, el denominador de la última expresión es el producto de todos los denominadores de la suma, por tanto, el grado del denominador (por tanto, la distancia entre las imágenes) es el mismo que el número de coordenadas que difieren de $b^{(1)}$ y $b^{(2)}$, que es su distancia de Hamming.

[2] Sólo hay que observar que, como los $c_j \in \mathbb{F}_2$, valen 0 ó 1, luego $f'(z) = \sum_{j:c_j=1} \prod_{k:k\neq j, c_k=1} (z - \gamma_k)$, que es el numerador resultante sacar común denominador en la suma de fracciones de $\xi(z)$.

3.2.2 Un criptosistema: el sistema de McEliece

El criptosistema de clave pública de McEliece fue presentado en 1978 ([18]), 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 3.3). 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, [17]).

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 1.1 (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 3.2. *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 1.1.*

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 3.16):

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

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 [18].

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

4

Conclusiones

A través del estudio de la computación cuántica y de la reducción del problema de la factorización al problema de encontrar el orden de un elemento módulo N hecha en el capítulo 2.1, llegamos a la conclusión de que era posible factorizar cualquier número natural en una cantidad razonable de tiempo, viendo finalmente un ejemplo sencillo para $N = 15$. A priori, ya que muchos otros sistemas no se basan en el problema de la factorización, como DSA, ECDSA o el criptosistema ElGamal (basados en el problema del logaritmo discreto), se puede pensar que estos sistemas resisten la llegada de los ordenadores cuánticos. Sin embargo, el problema de encontrar el orden puede generalizarse a uno más general, el del subgrupo oculto (HSP). Dicho problema puede formularse de la siguiente manera ([19]):

Definición 4.1. El problema del subgrupo oculto (HSP)

Dado un grupo G y una función $f : G \to S$, donde S es un conjunto finito que satisface:

1. Existe un subgrupo $H \leq G$ tal que f es constante en cada conjunto de la forma $gH, g \in G$.
2. Para cada $g, g' \in G$, $f(g) = f(g')$ si y sólo si $gH = g'H$.

El problema del subgrupo oculto (HSP) consiste en calcular H.

En el problema de la factorización, $G = (\mathbb{Z}, +)$, y $f : G \to (\mathbb{Z}/N\mathbb{Z})^*$ está definida por $f(x) = a^x \mod N$, con $a \in (\mathbb{Z}/N\mathbb{Z})^*$. Esta función verifica que es constante para cada conjunto de la forma $s + r\mathbb{Z}$, con r el orden del elemento a dentro del grupo multiplicativo $(\mathbb{Z}/N\mathbb{Z})^*$ por definición de orden ($a^{s+kr} \equiv a^s \mod N$); de aquí es sencillo ver que $f(s + kr) = f(s' + k'r)$ si y sólo si $s + r\mathbb{Z} = s' + r\mathbb{Z}$. Por tanto, la solución de este problema es $H = r\mathbb{Z}$. El algoritmo para resolver el HSP en este caso, es el circuito para encontrar el orden visto en la sección 2.2.3 de este trabajo, que proporciona el generador de H, r.

Para otros problemas, existen problemas HSP asociados. Algunos de ellos se resumen en la siguiente tabla, extraída de [4].

Grupo G	**Problema asociado**	**¿Algoritmo?**
Los enteros, $\mathbb{Z}$	Factorización	Sí
Grupos cíclicos finitos	Logaritmo discreto	Sí
Los reales, $\mathbb{R}$	Ecuación de Pell[3]	Sí
El grupo diédrico D_n, $n \geq 3$	Vector más corto de un retículo	No[1]
El grupo simétrico S_n, $n \geq 3$	Isomorfismo de un grafo	No[2]

[1] El algoritmo más rápido conocido toma tiempo subexponencial.
[2] Hay evidencias de dificultad para resolverlo en ordenadores cuánticos.
[3] La ecuación de Pell es una ecuación diofántica de la forma $x^2 - dy^2 = 1$ con $d \in \mathbb{N}$ libre de cuadrados. Más información puede consultarse en [20].

Para aquellos casos en los que G es abeliano, existe un método estándar para resolver el problema usando ordenadores cuánticos, que es en gran medida una generalización del circuito para encontrar el orden que hemos presentado en este trabajo. Es por ello que todos aquellos sistemas basados en el logaritmo discreto, también están en peligro con la llegada de los ordenadores cuánticos. El problema de resolver el HSP para grupos no abelianos, como los dos últimos presentados en la tabla, es aún un problema abierto. Por esta razón, por ejemplo, surge la criptografía basada en retículos como algoritmos de criptografía poscuántica, ya que no existen aún algoritmos conocidos que puedan resolver los problemas en los que se basan eficientemente.

Una vez estudiado cómo se podían romper gran parte de los sistemas actuales, vimos otras alternativas que, al menos por el momento, resisten a los ordenadores cuánticos. De la

criptografía basada en funciones hash extraíamos sistemas de firma, primero un esquema de un solo uso, y posteriormente una generalización para poder permitir más firmas con un solo par de claves. Sin embargo, estos sistemas presentan una gran desventaja: el gran tamaño de las claves. En el caso del sistema de firma basdo en árboles hash de Merkle estudiado en la sección 3.1.6, el poseedor de la clave privada debe almacenar 2^H pares de claves del sistema de un solo uso, además de toda la estructura de árbol completa. Si el sistema de un solo uso es LD-OTS, estos pares de claves tienen en total $2 \times 2n$ cadenas de n bits, es decir, $4n^2$ bits, con n el tamaño de la salida de la función hash utilizada. Por ejemplo, para $n = 256$, un par de claves ocuparía en total $4 \times 256^2 = 262{,}144$ bits, que son $32,768$ KB. Tomando un valor modesto $H = 8$, con lo que podríamos firmar $2^8 = 256$ veces, en total la clave privada del sistema MSS ocuparía más de 8 MB. En la práctica, es usual firmar muchas más veces (sobre todo en el caso de servidores web o en el caso de Autoridades de Certificación (CA)) y con un parámetro de seguridad n más elevado, lo cual hace que los tamaños de clave aumenten considerablemente.

Este problema puede resolverse en parte de dos maneras. En primer lugar, usando el sistema de firma de un solo uso de Winterneitz (W-OTS), también basado en funciones hash, se consiguen tamaños de clave más pequeños a cambio de realizar más evaluaciones de la función hash a la hora de generar el par de claves, con el respectivo aumento de tiempo (ver [4]). En segundo lugar, es posible evitar la necesidad de almacenar todos los pares de claves usando un generador de números pseudoaleatorios (PRNG). La idea en líneas generales es, en lugar de almacenar todas las claves, almacenar sólo una *semilla* para el generador, elegida de forma aleatoria. La aplicación sucesiva del generador a partir de la semilla inicial se usa para ir generando los pares correspondientes. La principal desventaja es que para acceder a un par en concreto, han de calcularse todos los anteriores; lo cual hace que el acceso a los pares de claves sea más lento en comparación con guardarlos todos en memoria [4].

A partir de la criptografía basada en códigos, obteníamos criptosistemas. La principal desventaja del sistema McEliece es, de nuevo, los tamaños de las claves: el sistema McEliece usa aproximadamente $n^2/4$ bits, siendo n el parámetro de seguridad del sistema. Para los niveles de seguridad comúnmente usados actualmente, una clave RSA puede ocupar unos pocos miles de bits; mientras que para una clave McEliece la longitud de la clave se

acerca al millón [4].

En definitiva, es por estas y por otras razones, que aún no hemos cambiado los sistemas criptográficos que usamos en la actualidad. La computación cuántica nos obliga a reexaminar los sistemas criptográficos que usamos. Algunos han sido rotos, y otros sistemas deben ser estudiados de nuevo para ajustar los tamaños de claves y parámetros que usan. Las alternativas que hay propuestas aún deben seguirse desarrollando y estudiando, de forma que cuando los ordenadores cuánticos sean una realidad, la comunidad criptográfica esté preparada para dar el salto a la criptografía poscuántica.

APPENDIX

A

Productos tensoriales

A.1 Definiciones y propiedades básicas

El producto tensorial de dos espacios vectoriales puede definirse de diversas maneras. Por ejemplo, siguiendo [8], una definición es la siguiente:

Definición A.1. Dados U, V dos espacios vectoriales de dimensión finita sobre un mismo cuerpo $\mathbb{K}$, se define el producto tensor, y denotado $U \otimes V$, al espacio dual del espacio vectorial formado por todas las formas bilineales en $U \oplus V$.

Para cada $x \in U$ e $y \in V$, el producto tensorial $z = x \otimes y \in U \otimes V$ está definido como la aplicación tal que $z(w) = w(x, y)$ para cada forma bilineal w de $U \oplus V$.

A partir de esta definición, es fácil ver las siguientes propiedades:

Proposición A.1. *Con la notación anterior, para cada $x_1, x_2 \in U$, $y_1, y_2 \in V$, $\lambda \in \mathbb{K}$, se verifica que:*

1. $(x_1 + x_2) \otimes y_1 = x_1 \otimes y_1 + x_2 \otimes y_1$.
2. $x_1 \otimes (y_1 + y_2) = x_1 \otimes y_1 + x_1 \otimes y_2$.
3. $\lambda(x_1 \otimes y_1) = (\lambda x_1) \otimes y_1 = x_1 \otimes (\lambda y_1)$.

Demostración.

1. Denotemos $z = (x_1 + x_2) \otimes y_1$. Por definición, para cada forma bilineal w, se verifica que $z(w) = w(x_1 + x_2, y_1)$. Aplicando bilinealidad en w, entonces $z(w) = w(x_1, y_1) + w(x_2, y_2)$. Denotando ahora $z_1 = x_1 \otimes y_1$ y $z_2 = x_2 \otimes y_1$, entonces tenemos que $z(w) = z_1(w) + z_2(w)$. Como esto se verifica para cada forma bilineal w, entonces $z = z_1 + z_2$, como queríamos ver.

2. Esta igualdad es análoga al caso anterior, aplicando bilinealidad en la segunda componente.

3. Denotando $z = (\lambda x_1) \otimes y_1, z' = x_1 \otimes y_1$, entonces para cada forma bilineal w, se tiene que $z(w) = w(\lambda x_1, y_1) = \lambda w(x_1, y_1) = \lambda z'(w)$. A partir de aquí, $\lambda z' = z$, como queríamos ver. Análogamente se puede ver, aplicando bilinealidad en la segunda componente, que $\lambda(x_1 \otimes y_1) = x_1 \otimes (\lambda y_1)$, lo cual concluye la prueba.

□

Nuestro objetivo, ahora, es determinar una base algebraica de $U \otimes V$ (y con ello, su dimensión). Para ello, necesitamos dos lemas previos sobre formas bilineales ([8]).

Lema A.1. *Sea U un espacio vectorial de dimensión n sobre un cuerpo $\mathbb{K}$ con base $\{u_1, \ldots, u_n\}$, y V un espacio vectorial de dimensión m sobre el mismo cuerpo $\mathbb{K}$, con base $\{v_1, \ldots, v_m\}$. Si $\{\alpha_{ij}\}$ es cualquier conjunto de escalares, $i = 1, \ldots, n$, $j = 1, \ldots, m$, entonces existe una única forma bilineal w en $U \oplus V$ tal que $w(u_i, v_j) = \alpha_{ij}$ para cada i, j.*

Demostración. Sean $u \in U$, $u = \sum_{i=1}^{n} \alpha_i u_i$, $v \in V$, $v = \sum_{j=1}^{m} \beta_j v_j$. Una forma bilineal w en $U \otimes V$ verifica que $w(u_i, v_j) = \alpha_{ij}$ si y sólo si:

$$w(u, v) = \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \beta_j w(u_i, v_j) = \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \beta_j \alpha_{ij}, \qquad \text{para cada } u \in U, v \in V.$$

Definiendo w a partir de esta igualdad se hace clara la existencia, y viendo que esta definción es una condición necesaria para que w verifique la propiedad expuesta, se tiene la unicidad. □

Lema A.2. *Sea U un espacio vectorial de dimensión n sobre un cuerpo $\mathbb{K}$ con base $\{u_1, \ldots, u_n\}$, y V un espacio vectorial de dimensión m sobre el mismo cuerpo $\mathbb{K}$, con base $\{v_1, \ldots, v_m\}$. Entonces, una base del espacio vectorial de todas las formas bilineales sobre $U \oplus V$ es $\{w_{pq}\}$, $p = 1, \ldots, n$, $q = 1 \ldots, m$, verificando que $w_{pq}(u_i, v_j) = \delta_{ip}\delta_{qj}$ (donde δ_{ij} representa la delta de Kronecker, que vale 1 si $i = j$ y 0 en caso contrario).*

Demostración. La existencia de dichos w_{pq} se deduce del lema anterior. Estos elementos son linealmente independientes, puesto que

$$\sum_{p=1}^{n}\sum_{q=1}^{m} \alpha_{pq} w_{pq} = 0$$

Implica que, evaluando en (u_i, v_j) para $i = 1, \ldots, n$,$j = 1, \ldots, m$:

$$0 = \sum_{p=1}^{n}\sum_{q=1}^{m} \alpha_{pq}\delta_{ip}\delta_{jq} = \alpha_{ij}$$

Sea ahora w una forma bilineal arbitraria de $U \oplus V$, y denotemos por $\alpha_{ij} = w(u_i, v_j)$, $i = 1, \ldots, n$,$j = 1, \ldots, m$. Veremos que $w = \sum_{p=1}^{n}\sum_{q=1}^{m} \alpha_{pq} w_{pq}$. En efecto, si $u = \sum_{i=1}^{n} \alpha_i u_i \in U$, $v = \sum_{j=1}^{m} \beta_j v_j \in V$, entonces

$$w_{pq}(u, v) = \sum_{i=1}^{n}\sum_{j=1}^{m} \alpha_i\beta_j\delta_{ip}\delta_{jq} = \alpha_p\beta_q$$

Y por tanto,

$$w(x, y) = \sum_{i=1}^{n}\sum_{j=1}^{m} \alpha_i\beta_j\alpha_{ij} = \sum_{i=1}^{n}\sum_{j=1}^{m} \alpha_i\beta_j w_{pq}(x, y)$$

□

Esto prueba que la dimensión del conjunto de todas las formas bilineales de $U \oplus V$ es mn, el cardinal de la base hallada. De aquí es inmediato que el producto tensorial de dos espacios vectoriales U y V también tiene dimensión mn, puesto que es el dual del espacio anterior. Podemos precisar y dar una base de $U \otimes V$.

Teorema A.1. *Si $\{x_1, \ldots, x_n\}$, $\{y_1, \ldots, y_m\}$ son bases de dos espacios vectoriales U y V respectivamente, entonces el conjunto $\{z_{ij} = x_i \otimes y_j\}$, $i = 1, \ldots, n$, $j = 1, \ldots, m$ es una base de $U \otimes V$.*

Demostración. Consideremos la base $\{w_{pq}\}$ del espacio de las formas bilineales en $U \oplus V$, con $p = 1, \ldots, n$, $q = 1 \ldots, m$ definida como en el Lema A.2. Consideremos su base dual $\{w'_{pq}\}$, de manera que $w'_{pq}(w_{ij}) = \delta_{ip}\delta_{iq}$ (es decir, vale 1 si $i = p$,$j = q$ y 0 en cualquier otro caso). Si $w = \sum_{p=1}^{n} \sum_{q=1}^{m} \alpha_{pq} w_{pq}$ es una forma bilineal arbitraria en $U \oplus V$, entonces

$$w'_{ij}(w) = \sum_{p=1}^{n} \sum_{q=1}^{m} \alpha_{pq} w'_{ij}(w_{pq}) = \sum_{p=1}^{n} \sum_{q=1}^{m} \alpha_{pq} \delta_{ip} \delta_{iq} = \alpha_{ij} = w(x_i, y_j) = z_{ij}(w)$$

De aquí, $z_{ij} = w'_{ij}$ y puesto que w'_{ij} forma una base de $U \otimes V$ por construcción, también los elementos $x_i \otimes y_j$. □

A.2 Aplicaciones lineales entre productos tensoriales de espacios

Fijemos dos espacios vectoriales U, V. ¿Qué forma tienen las aplicaciones lineales $f : U \otimes V \to U \otimes V$? Sean $x \in U$, $y \in V$ y $g : U \to U$, $h : V \to V$ dos aplicaciones lineales. Entonces, de manera natural, podemos definir una aplicación $g \otimes h : U \otimes V \to U \otimes V$ como

$$(g \otimes h)(x \otimes y) = g(x) \otimes h(y) \tag{A.1}$$

A partir de la linealidad del producto tensorial y de A y B, es trivial ver que $A \otimes B$ es lineal. La definición de esta aplicación es análoga si las aplicaciones lineales A y B no son endomorfismos.

Teorema A.2. *Con la notación anterior, si A y B son las matrices asociadas de g y h respectivamente, entonces la matriz asociada al operador $g \otimes h$ viene dada por la matriz*

$$A \otimes B = \begin{pmatrix} A_{11}B & A_{12}B & \ldots & A_{1n}B \\ A_{21}B & A_{22}B & \ldots & A_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1}B & A_{m2}B & \ldots & A_{mn}B \end{pmatrix}$$

En la notación anterior, $A_{ij}B$ denota el producto de la entrada (i, j) de la matriz A por la matriz B. La matriz $A \otimes B$ está definida, como puede verse, por bloques; si A es una matriz $m \times n$ y B es una matriz de dimensión $p \times q$, $A \otimes B$ es una matriz de dimensión $mp \times nq$. A este producto $\otimes$ entre matrices se le denomina *producto de Kronecker*. La demostración de este teorema puede hacerse a partir de la ecuación (A.1), evaluando $g \otimes h$ por los elementos de la base de $U \otimes V$.

APPENDIX

B

Código de los programas

B.1 Implementación del algoritmo de Shor en Mathematica

Aquí presentamos las principales dos funciones implementadas en Mathematica, que realizan las operaciones de las puertas $U_{f_{a,N}}$ y QFT del algoritmo de Shor, estudiado en la sección 2.2.3. El resto del algoritmo particularizado para $N = 15$ puede consultarse en el archivo `shor.nb`.

B.1.1 Puerta U_f

```
    Uf[estado_, a_, N_, m_, n_] := Module[{final = 0},
    For[i = 0, i <= 2^m - 1, i++,
        For[j = 0, j <= 2^n - 1, j++,
            final += Coefficient[estado, TensorProduct[Ket[i], Ket[j]]]*
    TensorProduct[Ket[i], Ket[Mod[j + PowerMod[a, i, N], 2^n]]];
        ];
    ];
    Return[final];
];
```

B.1.2 Puerta QFT

```
QFT[estado_, m_] := Module[{final = 0},
    For[j = 0, j <= 2^m - 1, j++,
        final += 1/Sqrt[2^m]* Coefficient[estado, Ket[j]]*Sum[Exp[(2 \[Pi
    ] I j k )/(2^m)]*Ket[k], {k, 0, 2^m - 1}];
```

```
    ];
    Return[final];
];
```

B.2 Implementación de la firma LD-OTS en Python

Aquí presentamos las principales clases y algoritmos implementados en Python usadas en la sección 3.1.4. Las claves públicas y privadas, junto con las firmas generadas se pueden encontrar en la carpeta adjunta. Las claves públicas tienen extensión `.PUK`, las claves privadas, `.PRK`; y las firmas de los documentos son aquellos ficheros acabados en `.SIG`.

B.2.1 Esquema de firma

Esta es una implementación del sistema de firma de un solo uso de Lamport-Diffie. Las funciones f y H y el parámetro n del esquema se pasan como parámetros. En los experimentos posteriores, hemos usado como f y H la función $SHA-256$ truncada a los primeros `n//8` bytes (n primeros bits si n es múltiplo de 8).

```
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!")
```

B.2.2 Generador de mensajes

Para poder generar un conjunto S de mensajes falsificados, normalmente se suele proceder de la siguiente manera: se elabora un texto base para falsificar, y se van insertando sinónimos entre algunas palabras. Así, eligiendo una palabra de entre cada par de sinónimos a lo largo del texto, se van obteniendo documentos que son equivalentes, pero con imágenes de la función hash distintas. El documento de ejemplo que se ha usado puede consultarse con más detalle en el archivo `crack.py`. Esta clase hace todo el trabajo de ir iterando por las posibles opciones a través de un iterador de Python.

```
class TextoVariable:

    def __init__(self,numpos,baseStr):
        self.bstr = baseStr
        self.n = numpos
        self.options = []
        for k in range(numpos):
            self.options.append((2,("","")))
        self.curr = [0 for k in range(self.n)]

    def setOption(self,index,options):
        self.options[index] = (len(options),options)

    def nextString(self):
        for k in range(self.n):
            if self.curr[k] < (self.options[k][0] - 1):
                self.curr[k] += 1
                break
            else:
                self.curr[k]=0
                if k==self.n - 1:
                    return ""

        ret = self.bstr
        for k in range(self.n):
            ret = ret.replace("~", self.options[k][1][self.curr[k]],1)
        return ret

    def iterator(self):
        str = self.nextString()
        while (str!=""):
            yield str
            str = self.nextString()
    def reset(self):
        self.curr = [0 for k in range(self.n)]
```

B.2.3 Algoritmo de búsqueda por fuerza bruta

Esta es la implementación del Algoritmo 3.2.

```
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))
```

B.2.4 Algoritmo de búsqueda con 2 firmas

Esta es la implementación del Algoritmo 3.3.

```
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. 1, 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. 1

[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. 1

[4] Daniel J. Bernstein and Johannes Buchmann And Erik Dahmen, editors. *Post-Quantum Cryptography*. Springer, 2009. 2, 7, 62, 63, 64

[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. 2

[6] Daniel J. Bernstein. Grover vs. McEliece. In Nicolas Sendrier, editor, *Post-Quantum Cryptography*, pages 73–80, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. 2, 58, 60

[7] Michael A. Nielsen and Isaac L.Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000. 7, 8, 18, 19, 20, 29, 30

[8] Paul R. Halmos. *Finite-Dimensional Vector Spaces*. Springer, 1916. 65, 66

[9] A. Aizpuru Tomás. *Apuntes incompletos de Análisis Funcional*. Servicio de Publicaciones de la Universidad de Cádiz, 2009. 15

[10] Noson S. Yanofsky and Mirco A. Mannucci. *Quantum computing for computer scientists*. Cambridge University Press, 2008. 17

[11] 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. 35, 44

[12] 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. 35

[13] 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. 35

[14] Douglas R. Stinson. *Cryptography, Theory and Practice*. Discrete Mathematics And Its Applications. Chapman & Hall/CRC, 3 edition, 2006. 35

[15] Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO'89 Proceedings*, pages 218–238, New York, NY, 1990. Springer New York. 48

[16] J. H. van Lint. *Introduction to Coding Theory*. Number 86 in Graduate Texts in Mathematics. Springer-Verlag Berlin Heidelberg, 1999. 53, 54

[17] N. Patterson. The algebraic decoding of Goppa codes. *IEEE Transactions on Information Theory*, 21(2):203–207, 1975. 57, 58

[18] R. J. McEliece. A public-Key cryptosystem based on algebraic coding theory. *DSN Progress Report*, 42-44:114–116, 1978. 58, 59

[19] R. De Wolf. Quantum computing: Lecture notes. *arXiv preprint arXiv:1907.09415*, 2019. 61

[20] Edward J. Barbeau. *Pell's Equation*. Problem books in mathematics. Springer, 2003. 62