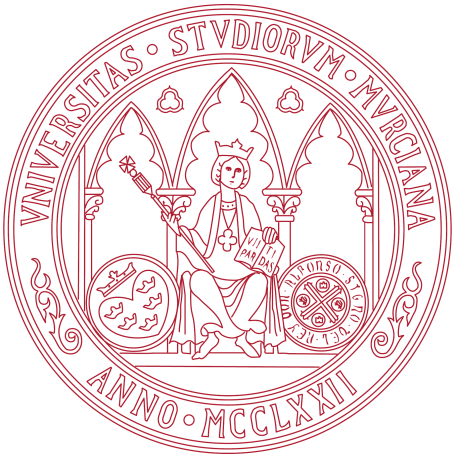
PRUEBAS DE CONOCIMIENTO CERO Y SUS APLICACIONES

JOSÉ LUIS CÁNOVAS SÁNCHEZ

Tutores
ANTONIO JOSÉ PALLARÉS RUIZ
LEANDRO MARÍN MUÑOZ



Facultad de Matemáticas
Universidad de Murcia

DECLARACIÓN DE ORIGINALIDAD

Yo, José Luis Cánovas Sánchez, autor del TFG PRUEBAS DE CONOCIMIENTO CERO Y SUS APLICACIONES, bajo la tutela de los profesores Antonio José Pallarés Ruiz y Leandro Marín Muñoz, declaro que el trabajo que presento es original, en el sentido de que ha puesto el mayor empeño en citar debidamente todas las fuentes utilizadas.[1]

Murcia, Julio 2017

José Luis Cánovas Sánchez

1 En la Secretaría de la Facultad de Matemáticas se ha presentado una copia firmada de esta declaración.

EXTENDED ABSTRACT

Nowadays, to proof someone we know a certain secret, we shall use a transmission channel where nobody could spy on us, and then we send the secret directly to our listener. In the scope of cryptography, we would cipher the secret with a key, in a symmetric or asymmetric scheme, such that only those who know the deciphering key can actually read our encrypted secret. This is the foundation of the Internet's security. We cipher our password in a way only the mail server can read it, so it deciphers the password and if it matches the one stored in its side, the server lets you log in and access to your account, without any eavesdropper stealing our password in between. The problem here is that the password is only secured during the transmission, two entities know the secret and therefore we have two possible points of attack where a malicious entity would not need to break de ciphering scheme used in the transmission channel.

There exists a special area in cryptology in charge of studying this issue, proving someone we know something, but that from the proof performed, no other information is obtained, except that the proof itself is correct. These methods are known as Zero-Knowledge Proofs, because they leak no extra knowledge from performing them. We chose to study these techniques due to the Computer Science degree thesis, *Integration of Idemix in IoT environments*, where Idemix is a cryptographic protocol based on zero-knowledge proofs. When we started the study of said thesis, we noticed the amount of mathematical results related, for this reason, we were interested in studying it deeply and formally, and considered it for this Mathematics degree thesis.

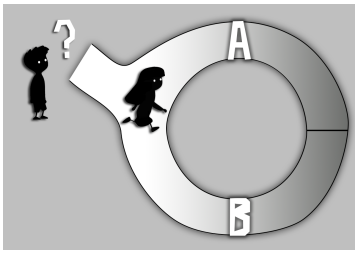
To illustrate how zero-knowledge proofs work, in 1990 Guillou, Quisquater and Berson published in *How to Explain Zero-Knowledge Protocols to Your Children* [12] a story about how Ali Baba proved he was able to open the door of the cave, without revealing anyone the magic words. Due to it being a 4 pages long story, here we present a brief version which highlights the basic properties of these methods:

> Imagine a cave, where the path forks in two passages, and at the end of each one, they join again, with the shape of a ring. In the point the paths meet, there is a magic door, that only opens when someones pronounces the magic work.
>
> **P**eggy knows the secret word and wants to **p**rove it to her friend, **V**ictor, but without revealing it. Peggy and Victor meet at the entrance of the cave, then Victor awaits while Peggy goes inside the cave, taking one of the passages, that we will name A and B. Victor can't see which way Peggy went.
>
> When Peggy arrives at the door, **V**ictor enters the cave, and when he arrive to the fork, stops and yells which path, A or B, he wants Peggy to come back, to **v**erify she knows how to open the door.
>
> If Peggy actually knows the secret, she always can take the requested path, opening the magic door if needed. But if Peggy doesn't know the magic word, she had a chance of 50% to guess correctly what passage Victor was going to ask. That means she had a chance to fool Victor.

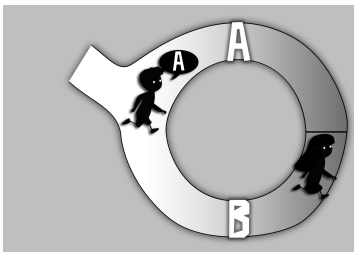
Victor then asks to repeat the experiment. With 20 repetitions, the chances Peggy fools Victor in all of them is only 2^{-20}, practically negligible, of guessing every time and therefore fool Victor.

Eve, curious about what Victor and Peggy were doing in the cave, **e**avesdrops Victor during the process. The problem is that Eve doesn't know if Peggy and Victor agreed on what paths to choose, because they wanted to prank her for being busybody. Only Victor is confident he is choosing the returning passage in a random way.

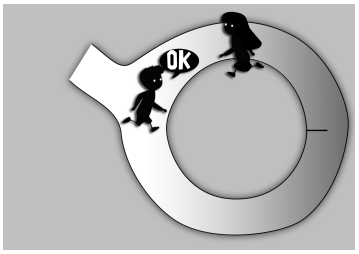
Later, Eve asks Victor about what happened in the cave. Victor tells Eve about the door and that he is convinced that the door can be opened with the magic word that Peggy knows, but he can't prove it to Eve because he can't open the door.



The cave [13]. Peggy takes randomly A or B. Victor awaits outside.



Victor chooses randomly the returning path for Peggy.



Peggy returns by the requested path.

Many different types of problems are used in the formal study of Zero-Knowledge Proofs, in which those related to number theory and graph theory stand out the most. Theses problems are hard to solve for those who don't know any extra information about the problem, in our story the problem would be opening the door without knowing the magic word.

The most representative problems in Zero-Knowledge Proofs are the discrete logarithm, the square root modulo a composite integer, the graph isomorphism, finding hamiltonian cycles and the 3-colorability of graphs.

The discrete logarithm consists on, given a cyclic group of prime order, a generator g and an element y in that group, finding the power s such that $g^s = y$, that is, the discrete logarithm base g of y, $\log_g y$.

The problem of the modular square root involves the quadratic residues theory, which shows us that if the modulus is a prime number, it is easy to find the square root, if it exists, of a given value, but if the modulus is composite, and we don't know its factorization in prime factors, it is hard to even tell if a given value is a quadratic residue, that is, it has square roots.

The graph isomorphism problem consists on given two distinct graphs, tell if there exists an isomorphism between them. This problem has many particular cases easy to solve, but the best known algorithm to solve any instance of the problem is only quasi-polynomial in time, and it is still under revision.

A hamiltonian graph is that with a cycle with every node in the graph, without repetitions.

The 3-colorability of a graph consists on giving each pair of connected nodes two different colors, in a way all the nodes in the graph have only one color, and we used exactly 3 colors.

The hamiltonian cycle and 3-colorability problems have in common that both are categorized in the class of problems known as **NP**-complete. This means that solving one **NP**-complete problem is equivalent to solve any other, but to this day no efficient algorithm for any of them is known.

We divide this paper in three parts, during the first one we will study all these problems, starting with defining what we understand a *hard* problem actually is, from a computational point of view, then we will formally describe the mentioned problems, as well as the algorithms that, knowing some extra information (the secret), allow us to solve them.

The second part of this memory will focus on the Zero-Knowledge Proofs themselves, defining what properties must have a proof of this kind, and we will state various proofs related to the problems previously described, proving that each one of them satisfy the definition.

We will first define what an interactive proof formally is, so we then can build on top of it the definition of a Zero-Knowledge Proof, which intuitively is interactive. The first kind of Zero-Knowledge Proof is known as perfect, because no machine would be capable of obtaining any extra information from them. Because this particular definition is very constraining, we will relax it in two ways, the first one, by supposing that our verifier is not malicious, that is, while we interact with it, the verifier won't try to extract extra information from us. The last type of Zero-Knowledge Proofs will be those called computational, in which instead of considering a perfect security against any machine, we consider real world machines, limited in time and memory, and in practice it will give us as much security as the perfect Zero-Knowledge Proofs.

The third part will be dedicated to the practical applications of Zero-Knowledge Proofs. The use of these protocols will let us solve problems related to privacy during communications, increasing application's security and functionality.

One of the most important results in this chapter is the non-interactivity of Zero-Knowledge Proofs, through the Fiat-Shamir heuristic, which uses hash functions, hard to predict in the practice.

Combining many techniques of the Zero-Knowledge Proofs, IBM developed a security suite protocol for privacy preserving, called Identity Mixer. The issuance of credentials uses the Camenisch-Lysyanskaya signature, which most strong features are that it can be performed between two parties, and it can be randomized, in a way some values of the signature change affected by a random parameter, but the signature is still valid.

Then, a certificate can be partially disclosed, that is, when a user presents its certificate to a verifier, only part of the certificate is shown, and the other part is committed with a Zero-Knowledge Proof assuring the hidden values are signed by a valid Camenisch-Lysyanskaya signature, which is also randomized to avoid correlations between different uses of the certificate.

Finally, we include an appendix to show some implementations of these algorithms using `sage`, a Python based mathematical software of free distribution. It offers tools for modular arithmetic so we can focus on the implementation of the protocols instead of writing efficient implementations of modular arithmetic operations to work in other programming languages. These implementations are not related to the Computer Science thesis, which uses the C programming language and focuses on other kind of devices.

ÍNDICE GENERAL

1 INTRODUCCIÓN

En la actualidad, para demostrar que conocemos un secreto a alguien, utilizamos un medio donde nadie nos espíe y le contamos el secreto directamente a nuestro interlocutor. En el ámbito de la criptografía, cifraríamos el secreto con una clave, simétrica o asimétrica, tal que, sólo quien conozca la clave de descifrado pueda leer nuestro secreto. Esto es la base de las comunicaciones seguras en Internet. Ciframos nuestra contraseña de modo que sólo el servidor de correo pueda leerla, pudiendo iniciar nuestra sesión sin que ningún espía nos la pueda robar. El problema es que hay dos partes que conocen el secreto, dos puntos de ataque.

Existe un área de la criptología que se encarga de abordar este problema, demostrar que conocemos algo, pero sin que de la misma prueba se obtenga más información. Estos métodos se conocen como pruebas de conocimiento cero. Elegimos estudiar estas técnicas a raíz del TFG de Ing. Informática, *Integración de Idemix en entornos IoT*, donde Idemix es un protocolo criptográfico basado en pruebas de conocimiento cero. Al inicio del estudio de dicho trabajo, observamos la gran cantidad de resultados matemáticos relacionados, por ello, nos interesó estudiarlo de manera formal y en profundidad, y lo enfocamos como Trabajo de Fin de Grado de Matemáticas.

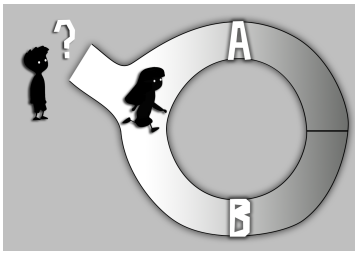
Para ilustrar las pruebas de conocimiento cero, en 1990, Guillou, Quisquater y Berson publicaron en *How to Explain Zero-Knowledge Protocols to Your Children* [12] una historia sobre cómo Alí Babá demostró poder abrir la cueva sin decir a nadie cuáles eran las palabras mágicas. Aquí presentamos una versión resumida que destaca las propiedades básicas de estos métodos:

> Imaginemos una cueva, donde el camino principal se bifurca y al final de cada pasillo los caminos se vuelven a encontrar, formando una especie de anillo. En el punto en que se unen, dentro de la cueva, hay una puerta mágica con una palabra secreta, la cual permite abrirla y cruzar al otro lado.
>
> **P**aula conoce la clave secreta y quiere **p**robarlo a su amigo Víctor, pero sin tener que revelársela. Paula y Víctor quedan en la entrada de la cueva con unos *walkie-talkies*, de modo que Víctor esperará fuera y Paula entrará a la cueva y tomará uno de los pasillos, que llamaremos A y B, sin decirle cuál a Víctor.
>
> Al llegar a la puerta, Paula avisa a Víctor para que entre a la cueva y espere en la bifurcación, donde **V**íctor, para intentar **v**erificar que Paula conoce la palabra secreta, le indicará por qué pasillo quiere que vuelva, el A o el B.
>
> Si Paula realmente conoce el secreto, podrá volver a la bifurcación por el pasillo solicitado, abriendo, si es preciso, la puerta.
>
> Pero en caso de no conocer la clave, al entrar por uno de los pasillos, tenía una probabilidad del 50 % de adivinar cuál pediría Víctor.
>
> Víctor no se queda contento con una sola prueba, pues Paula podría haber tenido suerte, así que la repiten hasta que se convence. Con unas 20 repeticiones,

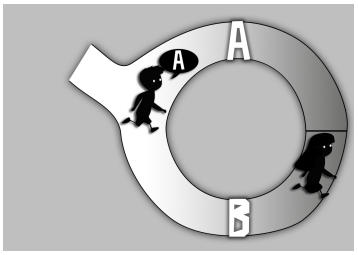
Paula tendría solo una probabilidad de 2^{-20}, prácticamente nula, de acertar todas las veces y engañar así a Víctor.

Eva, curiosa de qué hacían Víctor y Paula en la cueva, **e**spía a Víctor durante todo el proceso. Eva no sabe si Paula y Víctor han acordado previamente qué pasillo pedir por el *walkie-talkie*, y sólo Víctor está seguro de que él los estaba eligiendo al azar y sin previo acuerdo.

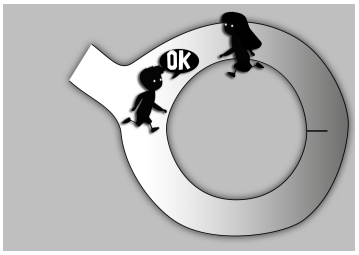
Más tarde Eva habla con Víctor, que está seguro de que Paula conoce la clave, y éste querría convencer también a Eva, pero como él no conoce la clave, no puede repetir la prueba a Eva, sólo Paula puede realizarla con éxito.



La cueva [13]. Paula entra por A o B al azar. Víctor espera fuera.



Víctor elige al azar por dónde quiere que regrese Paula.



Paula vuelve por el camino pedido.

En el estudio formal de las Pruebas de Conocimiento Cero se utilizan distintos tipos de problemas, y los más utilizados están relacionados con la **teoría de números** y la **teoría de grafos**. Son problemas difíciles de resolver para quien no conoce alguna información extra de los mismos, que en nuestra fábula serían el problema de abrir la puerta sin conocer la palabra mágica. Los problemas más representativos de las Pruebas de Conocimiento Cero son el del logaritmo discreto, la raíz cuadrada modular, el isomorfismo de grafos, el cálculo de caminos hamiltonianos en grafos y la 3-coloración de grafos.

La memoria se divide en tres partes, en la primera parte estudiaremos todos estos problemas, empezando por definir qué se entiende por un problema *difícil* desde el punto de vista de la computación, para después describir formalmente los problemas mencionados anteriormente así como los algoritmos que nos permiten resolverlos cuando disponemos de una información adicional (el secreto).

La segunda parte de la memoria se dedicará a las Pruebas de Conocimiento Cero en sí mismas, definiremos qué propiedades debe cumplir una prueba de este tipo, enunciaremos pruebas que utilicen los problemas anteriores, y demostraremos que cada una de ellas cumple la definición.

La tercera parte de la memoria estará dedicada a las aplicaciones prácticas de estas Pruebas de Conocimiento Cero. La utilización de estos algoritmos permite resolver problemas relacionados con la privacidad de las comunicaciones, aumentando la seguridad y la funcionalidad de las aplicaciones.

Por último, dedicaremos un apéndice a mostrar algunas implementaciones de estos algoritmos utilizando `sage`, que es un software matemático basado en Python de libre distribución. Éstas no corresponden a las implementaciones realizadas durante el TFG de Ing. Informática, realizadas en C para otro tipo de dispositivos.

2

PROBLEMAS, ALGORITMOS Y COMPLEJIDAD

En este capítulo introducimos unos preliminares sobre la complejidad computacional, que nos permitirán entender qué problemas matemáticos, y por qué, se utilizan en criptografía. Comencemos con unas definiciones:

Definición 2.1. Llamamos *problema* a la descripción general de una tarea que depende de unos parámetros. La *definición* de un problema consta de dos partes. La primera da el escenario del problema, describiendo los parámetros necesarios. La segunda parte indica una pregunta de la que se espera una respuesta o solución.

Ejemplo 2.2. Vamos a considerar el problema de multiplicar dos matrices. La definición del problema sería:

Nombre:	Problema multiplicación de matrices.
Parámetros:	Dos matrices A_1 y A_2.
Pregunta:	¿Cuál es la matriz A tal que $A = A_1 \cdot A_2$?

Definición 2.3. Una *instancia* de un problema es el caso particular de un problema al que se le han dado valores a los parámetros.

Definición 2.4. Un *algoritmo* es una lista de instrucciones que para una instancia produce una respuesta correcta. Se dice que un algoritmo *resuelve* un problema si devuelve respuestas correctas para todas las instancias del problema.

Definición 2.5. Se llama *problema de decisión* a un problema cuya respuesta está en el conjunto $\mathcal{B} = \{\texttt{Verdadero}, \texttt{Falso}\}$.

Asociado a un problema, podemos establecer un problema de decisión que a veces puede resultar de una dificultad muy diferente del problema original. Consideremos el siguiente problema de decisión asociado al problema planteado del producto de matrices:

Ejemplo 2.6.

Nombre:	Problema de decisión de multiplicación de matrices.
Parámetros:	Tres matrices A, A_1 y A_2.
Pregunta:	¿$A = A_1 \cdot A_2$?

Un algoritmo para solucionarlo sería realizar el producto de las matrices A_1 y A_2 y comprobar que el resultado coincide con A. Esto nos daría una solución del problema por reducción al caso anterior y por lo tanto la dificultad del problema sería la misma que la del problema original.

Sin embargo, podemos encontrar en algunos casos algoritmos más eficientes que nos permitan resolver el problema de decisión pero que no establecen una solución al problema inicial.

Este ejemplo de la multiplicación de matrices, lo trataremos más adelante, pero se puede entender la diferencia fundamental que puede suponer la información adicional que nos ha ofrecido el nuevo problema con otro ejemplo. Mientras que dado un número, determinar

cuales son sus factores es un problema muy difícil, el problema de determinar si un factor concreto es un divisor de un número es tan simple como hacer una división.

Todos estos ejemplos los trataremos más en profundidad a lo largo de la memoria, pero antes de seguir adelante vamos a introducir más formalmente el concepto de complejidad.

2.1 NOTACIÓN ASINTÓTICA

Para poder estudiar cómo crecen el tiempo de ejecución, la memoria usada, o cualquier otro recurso, de un algoritmo, dependiendo del tamaño de los datos de entrada, utilizamos distintas *notaciones asintóticas*. Nos interesa en particular la siguiente:

Definición 2.7. Sea $f : \mathbb{N} \to \mathbb{R}^+$. Definimos la notación *O grande* u *orden* de f al conjunto de funciones de $\mathbb{N}$ a $\mathbb{R}^+$ acotadas superiormente por un múltiplo positivo de f a partir de cierto valor $n \in \mathbb{N}$:
$O(f) = \{t : \mathbb{N} \to \mathbb{R}^+ \mid \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} : t(n) \leqslant c \cdot f(n) \quad \forall n \geqslant n_0\}$.

Para estudiar el tiempo de ejecución, $t : \mathbb{N} \to \mathbb{R}^+$ (el tiempo promedio, el caso más desfavorable, ...), de un algoritmo, buscaremos una función f que acote a t lo más de cerca posible. Para ello definimos la relación de orden:

Definición 2.8. Sean $f, g : \mathbb{N} \to \mathbb{R}^+$. Diremos que $O(f) \leqslant O(g)$ si $\forall t \in O(f)$ se tiene que $t \in O(g)$, es decir, $O(f) \subseteq O(g)$.

Para acotar t buscaremos el menor $O(f) : t \in O(f)$.

2.2 ALGORITMOS POLINOMIALES, EXPONENCIALES Y SUBEXPONENCIALES

Antes de empezar a ver los tipos de algoritmos, es interesante ver un ejemplo. Pensemos en el algoritmo tradicional de suma de dos números decimales que por simplicidad consideraremos que tienen el mismo número de cifras l. Si repasamos este algoritmo, lo que tenemos que hacer es ir sumando una a una cada cifra de los dos números teniendo en cuenta si nos llevamos algo de la operación anterior. Esto nos da un número de operaciones que es esencialmente l, o $2l$ si consideramos también la corrección dependiente de la suma anterior.

Un algoritmo como este es muy efectivo puesto que aunque los números involucrados tengan 8 ó 10 cifras, un niño en la escuela puede hacerlo sin dificultad en un tiempo relativamente corto.

El algoritmo de la multiplicación ya es un poco más complicado, puesto que tenemos que multiplicar cada una de las cifras del primer número por cada una de las cifras del segundo y luego sumar, eso nos da un número de operaciones dependientes de l^2 y no de l como antes, pero aun así el algoritmo es fácil de hacer con papel y lápiz si no superamos las 4 ó 5 cifras.

Sin embargo, pensemos en el algoritmo para el cálculo del factorial, el número de operaciones ahora no depende de l, el número de cifras, sino de el valor del número, pues si estamos hablando de la base 10, serían 10^l el número aproximado de multiplicaciones que tendríamos que hacer. Esto sería totalmente imposible hacerlo con lápiz y papel para

valores tan pequeños como $l = 3$ porque requerirían un número entre 100 y 999 multiplicaciones.

Cuando el tiempo que tarde un algoritmo depende de una potencia del número de cifras de la entrada (lo que se suele llamar del tamaño de la entrada) diremos que el algoritmo es polinómico, en el ejemplo anterior, la suma depende de l y el producto de l^2. Pero si el tiempo depende del valor concreto del número como en el caso del factorial diremos que el algoritmo es exponencial.

Vamos a formalizar estos conceptos utilizando la notación asintótica. Para ello empecemos fijándonos que el número de cifras en base b de un número n es $\log_b(n) = \ln(n)/\ln(b)$ por lo que cambiar de base únicamente nos modifica una constante, que en la notación asintótica se desprecia, es decir $\log_b(n) \in O(\ln(n))$.

Podemos decir por tanto que el tamaño de un número es del orden $\ln(n)$ puesto que sea la que sea la base con la que trabajemos, la diferencia será una constante.

Definición 2.9. Se llama algoritmo de *tiempo de cálculo polinomial* al algoritmo cuya función de tiempo del caso más desfavorable es de orden $O(l^k)$, donde l es el tamaño de la entrada y k una constante. Si la entrada es numérica, tal y como hemos visto l se puede tomar $\ln(n)$. Cualquier algoritmo cuyo tiempo de ejecución no puede acotarse de esa manera se llama de *tiempo de cálculo exponencial*.

En los problemas que trataremos en esta memoria, aparecen algoritmos que no siendo polinómicos, tampoco son puramente exponenciales. Estos son conocidos como algoritmos subexponenciales y vendrán dados por un parámetro α que de forma contínua nos dará una cierta medida de la complejidad.

Definición 2.10. Sea α un número real en el intervalo $[0, 1]$ y sea c un número real positivo. Llamaremos función subexponencial de parámetros α y c a la función

$$L_n[\alpha, c] = e^{c(\ln n)^\alpha (\ln\ln n)^{1-\alpha}}$$

Esta notación tiene dos casos extremos, que son el caso $\alpha = 0$ que nos proporciona las funciones $L_n[0, c] = e^{c(\ln\ln n)} = (\ln n)^c$ y el caso $\alpha = 1$ que nos da $L_n[1, c] = e^{c(\ln n)} = n^c$. Por lo tanto, un algoritmo que nos de un tiempo de ejecución $O(L_n[0, c])$ es un algoritmo polinómico, mientras que uno que nos de un tiempo de ejecución $O(L_n[1, c])$ es puramente exponencial.

Estrictamente hablando, cuando $\alpha \neq 0$ la complejidad se considera exponencial, pero sin embargo este parámetro α puede tomar muchos valores intermedios y será útil para precisar la complejidad de algunos algoritmos.

2.3 CLASES DE COMPLEJIDAD

Tal y como hemos visto en la sección anterior, un algoritmo se denomina de complejidad *polinómica* cuando la función de tiempo del caso más desfavorable es de orden $O(l^k)$, donde l es el tamaño de la entrada y k una constante.

A veces se denominan *buenos* o *eficientes* a los algoritmos polinomiales, e *ineficientes* a los exponenciales, siendo los subexponenciales una solución intermedia. Estas propiedades dependen de la notación asintótica, sin embargo, hay que hacer algunas consideraciones. Por un lado están las constantes ocultas, aunque la notación asintótica nos diga que es lo mismo que la complejidad sea l^3 que $10^{20}l^3$, la constante oculta 10^{20} puede generar importantes diferencias. Por otro lado, especialmente en criptografía, los tamaños con los que

nos movemos no suelen variar demasiado, y debemos ser conscientes de que un algoritmo exponencial puede llegar a ser más eficiente que uno polinómico para valores pequeños de l.

A pesar de estas puntualizaciones que podríamos hacer en casos concretos y que trataremos a lo largo de esta memoria, podemos considerar que la notación asintótica es una buena medida para el estudio de la complejidad de los algoritmos.

Basándonos en esta notación asintótica, podemos definir las siguientes clases de complejidad en problemas de decisión:

Definición 2.11. El conjunto de problemas de decisión que se pueden resolver en tiempo polinomial se llama clase **P**.

Definición 2.12. Se llama clase de complejidad **NP** al conjunto de problemas de decisión donde una respuesta que sea $Verdadero$ se puede verificar en tiempo polinomial, dada cierta información extra, que llamaremos *certificado* o *testigo* [1].

Es inmediato que que $\mathbf{P} \subseteq \mathbf{NP}$.

Ejemplo 2.13 (Problema en **NP**)**.** Consideremos el siguiente problema de decisión:

Nombre:	Problema del entero compuesto.
Parámetros:	Un entero positivo n.
Pregunta:	¿Es n compuesto? Es decir, ¿existen enteros a, $b > 1$ tal que $n = ab$?

El problema pertenece a **NP** porque se puede comprobar en tiempo polinomial que n es compuesto si nos dan su *certificado*, un divisor a de n, tal que $1 < a < n$. Basta usar la división euclídea de n entre a y ver que el resto es 0. Sin embargo, aún no se conoce si el problema pertenece a **P**.

Definición 2.14. Sean dos problemas de decisión $L_1, L_2 \in \mathbf{NP}$, con correspondientes conjuntos de instancias I_1 e I_2. Sean I_1^+ e I_2^+ los subconjuntos de todas las instancias "$Verdaderas$" de I_1 e I_2, respectivamente. Decimos que L_1 es *reducible en tiempo polinomial* a L_2, $L_1 \leqslant_P L_2$, si existe una función $f : I_1 \Rightarrow I_2$ que cumple:

1. Es ejecutable en tiempo polinomial.
2. $x \in I_1^+ \Leftrightarrow f(x) \in I_2^+$.

De manera más informal, se dice que L_1 se reduce en tiempo polinomial a L_2 si hay un algoritmo que resuelve L_1 utilizando como subrutina un algoritmo que resuelve L_2, y que se ejecuta en total en tiempo polinomial, si lo hace el algoritmo que resuelve L_2.

Si $L_1 \leqslant_P L_2$, entonces se puede entender que L_2 es, al menos computacionalmente, tan difícil como L_1, o que L_1 no es más difícil que L_2.

Definición 2.15. Un problema de decisión L se dice que es **NP**-completo, o **NPC** si:

1 Utilizaremos certificado y testigo como traducción de *certificate* y *witness*. Dependiendo del contexto se suele usar un nombre u otro, incluso a veces se considera que dicha información adicional es el *secreto* al que se refieren las pruebas de conocimento cero.

(i) $L \in \mathbf{NP}$, y

(ii) $L_1 \leqslant_P L \quad \forall L_1 \in \mathbf{NP}$.

Los problemas **NPC** son los más difíciles entre los **NP** en el sentido de que son al menos tan difíciles como el resto de problemas en **NP**.

Una pregunta que nos puede aparecer es si realmente existen estos problemas de tipo **NPC**. En este caso la respuesta es afirmativa, dichos problemas existen y basta con encontrar uno de ellos. Veamos el problema **NPC** más característico:

Nombre:	Problema de satisfacibilidad booleana (SAT).
Parámetros:	Una colección finita C de expresiones booleanas con variables y sin cuantificadores.
Pregunta:	¿Hay alguna asignación de las variables que haga Verdadero a C?

Teorema 2.16 (Teorema de Cook (1971))**.** *El problema de satisfacibilidad booleana es **NPC**.*

O expresado de otra manera:

Teorema 2.17. *Todo problema* $Q \in \boldsymbol{NP}$ *se puede reducir en tiempo polinomial al problema de satisfacibilidad booleana:* $\forall Q \in \boldsymbol{NP}$, $Q \leqslant_P SAT$.

Existen otros problemas del tipo **NPC** que nos aparecerán más adelante, como el de la 3-coloración de grafos, pero la existencia de uno es suficiente por la equivalencia teórica de todos ellos.

A día de hoy conocemos ciertas propiedades sobre las clases de equivalencia, pero quedan muchas cuestiones sin resolver, entre ellas, uno de los siete problemas del milenio, ¿**P** = **NP**?

La opinión de los expertos en base a ciertas evidencias es que la respuesta a las tres es *no*, pero hasta que se encuentren demostraciones formales de cada una, quedarán en conjeturas. Y basándose en esas conjeturas es donde se apoya la seguridad informática.

Existen tres problemas que se conoce que pertenecen a **NP**, pero se desconoce a día de hoy si pertenecen a **P** o **NPC**: el isomorfismo de grafos, el logaritmo discreto y la factorización de enteros.

En los siguientes capítulos los estudiaremos como base de diferentes pruebas de conocimiento cero.

2.4 ALGORITMOS PROBABILÍSTICOS

Para intentar resolver la infactibilidad computacional de ciertos problemas, surge un modelo alternativo de computación que utiliza métodos probabilísticos. Estos métodos no pueden asegurar cotas superiores absolutas de tiempo, e incluso pueden devolver una respuesta errónea. Sin embargo, dadas unas cotas muy pequeñas de errores, en la práctica, ciertos métodos probabilísticos son más eficientes que los algoritmos conocidos, pues el tiempo de ejecución *esperado* del método, calculado probabilísticamente, es menor que el orden del algoritmo original.

Definición 2.18. Llamamos *algoritmo de Monte Carlo* a un algoritmo probabilístico que resuelve un problema de decisión, pero tiene un error ϵ de equivocarse.

Decimos que es *parcialmente* Verdadero si cuando se le da una instancia Verdadera nunca se equivoca, pero si la instancia es Falsa puede devolver Verdadero con probabilidad ϵ. De la misma manera, decimos que es *parcialmente* Falso si siempre resuelve correctamente instancias Falsas, pero puede cometer un error al resolver instancias Verdaderas.

Definición 2.19. Llamamos *algoritmo de Las Vegas* a un algoritmo probabilístico que resuelve un problema de decisión, pero o bien lo resuelve correctamente, o bien informa de error y termina sin resolver el problema con una probabilidad ϵ.

Un problema que es tratado en muchos textos como un problema probabilístico aunque hoy en día ya no es un ejemplo válido es el de la primalidad. Vamos a exponerlo porque también nos servirá para explicar algunos otros aspectos de la complejidad de algoritmos.

Ejemplo 2.20 (Test de primalidad)**.** El problema de decisión:

Nombre:	Problema de primalidad (PRIM).
Parámetros:	Un entero n.
Pregunta:	¿Es n primo?

Este problema ha sido tratado tradicionalmente como un problema de tipo **NP** aunque en el año 2002 se encontró un algoritmo determinista de tiempo polinómico que resolvía el problema. Este algoritmo se conoce como AKS por el nombre de sus descubridores, M. Agrawal, N. Kayal y N. Saxena. A pesar de ser un algoritmo polinómico, su compejidad teórica es del orden $O((\log n)^{12+\epsilon})$, ver [15, Página 314], lo cual es muy ineficiente para el tamaño de enteros que se utilizan en la práctica.

Existe sin embargo un algoritmo muy conocido de tipo probabilístico que nos determina si un entero es compuesto con total seguridad o primo con una probabilidad de error tan pequeña como deseemos, es el algoritmo de Miller-Rabin. Este algoritmo es el que se sigue usando para las aplicaciones prácticas puesto que el error que puede ofrecer puede hacerse tan pequeño que sea incluso menor que el de un error en el hardware.

Otro ejemplo que podemos considerar para ver la diferencia entre un algoritmo determinista y uno probabilístico es el del producto de matrices. Dadas dos matrices A_1 y A_2 determinar si una tercera matriz A es el producto de las dos anteriores.

Una solución al problema es hacer el producto y comprobar que el resultado es igual a A. Utilizando el algoritmo tradicional de multiplicación de matrices, la complejidad de este algoritmo sería n^3 si consideramos, para simplificar, matrices cuadradas de tamaño n. Existen algoritmos de multiplicación algo más eficientes, pero vamos a ver cómo se podría resolver el problema utilizando un algoritmo probabilístico.

Si el producto de dos matrices A_1A_2 es distinto de 0, sabemos que el núcleo de la aplicación lineal que representa dicho producto es de dimensión como máximo $n-1$, por lo que el número de vectores X en el núcleo es como máximo $|K|^{n-1}$ siendo $|K|$ el número de elementos del cuerpo. La probablidad de que el vector X esté en el núcleo si este producto de matrices es distinto de 0 es por lo tanto menor que $\frac{1}{|K|}$. Eligiendo aleatoriamente vectores $X_1, X_2, ..., X_t$ y comprobando que $A_1(A_2X_i) - A = 0$ (lo cual puede hacer con complejidad n^2 si se hace en este orden) podemos obtener un algoritmo probabilístico de complejidad cuadrática y con un error menor que $\frac{1}{|K|^t}$, valor que se puede hacer tan pequeño como queramos de forma bastante rápida.

3

PROBLEMAS BASADOS EN GRAFOS

3.1 INTRODUCCIÓN A LA TEORÍA DE GRAFOS

Definición 3.1. Un *grafo* (simple no dirigido) es un par (V, E) formado por un conjunto finito $V \neq \emptyset$, a cuyos elementos denominaremos *vértices* o *nodos*, y E, un conjunto de pares (no ordenados y formados por distintos vértices) de elementos de V, a los que llamaremos *aristas*.

A los nodos v_1 y v_2 que forman una arista $e = (v_1, v_2)$ se les llama *extremos* de e.

Definición 3.2 (Conceptos).

- A dos nodos v_1 y v_2 que forman parte de una misma arista se les llama *adyacentes*.
- Se dice que una arista es *incidente* con un vértice v cuando v es uno de sus extremos.
- El grado de un nodo $v \in V$ es el número de aristas incidentes en él.
- Un camino es una sucesión de nodos $(v_1, v_2, \cdots, v_k)$ tales que para todo $i \in \{1, \cdots, k-1\}$ se tiene que $(v_i, v_{i+1}) \in E$.
- La longitud de un camino se define como el número de aristas que lo forman, en el caso del camino $(v_1, v_2, \cdots, v_k)$ la longitud es $k-1$.
- Un camino $(v_1, v_2, \cdots, v_k)$ se dirá un ciclo si $v_1 = v_k$ y no se repite ningún otro nodo del grafo, es decir, para todo $i, j \in \{1, \cdots, k-1\}$, si $i \neq j$ entonces $v_i \neq v_j$.
- Un ciclo $(v_1, v_2, \cdots, v_k)$ se dirá un ciclo hamiltoniano si contiene a todos los nodos.
- Un grafo diremos que es hamiltoniano si tiene un ciclo hamiltoniano.

Definición 3.3. Dos grafos $G_1 = (V_1, E_1)$ y $G_2 = (V_2, E_2)$ se dice que son *isomorfos*, lo cual denotaremos como $G_1 \simeq G_2$, si existe una aplicación biyectiva $\tau : V_1 \to V_2$ tal que $\forall u, v \in V_1 \ (u, v) \in E_1 \Leftrightarrow (\tau(u), \tau(v)) \in E_2$. Tal aplicación recibe el nombre de *isomorfismo*.

Proposición 3.4. *Dados dos grafos isomorfos $G_1 = (V_1, E_1)$ y $G_2 = (V_2, E_2)$, entonces G_1 tiene un ciclo hamiltoniano si y sólo si G_2 tiene un ciclo hamiltoniano. Es decir, la propiedad de ser un grafo hamiltoniano se conserva por isomorfismos.*

Demostración. Sea $\tau : V_1 \to V_2$ el isomorfismo y sea $(v_1, v_2, \cdots, v_k)$ el ciclo hamiltoniano de V_1. Entonces $(\tau(v_1), \tau(v_2), \cdots, \tau(v_k))$ el el ciclo hamiltoniano de G_2 ya que, por un lado es un ciclo puesto que $(v_i, v_{i+1}) \in E_1$ implica que $(\tau(v_i), \tau(v_{i+1})) \in E_2$, y por otro, al ser τ biyectiva, el ciclo contiene a todos los nodos sin repetir ninguno (salvo los extremos). □

Definición 3.5. Una *coloración* de un grafo $G = (V, E)$ es una aplicación $\phi : V \to \{1, 2, \ldots, l\}$. El valor $\phi(v_i)$ es el *color* correspondiente al nodo v_i.

Definición 3.6. Una *coloración propia* de un grafo es una coloración que hace corresponder colores diferentes a vértices adyacentes, es decir $\forall (u, v) \in E$ se tiene que $\phi(u) \neq \phi(v)$.

Definición 3.7. Llamaremos *número cromático* del grafo G, $\chi(G)$, al mínimo valor de l que permite una coloración propia de G, es decir, al mínimo número de colores necesarios para colorear los vértices de forma que los extremos de cada arista tengan colores distintos.

En las siguientes secciones de este capítulo, introducimos los enunciados de los problemas de grafos utilizados en pruebas de conocimiento cero.

3.2 EL PROBLEMA DEL ISOMORFISMO DE GRAFOS

Nombre:	Problema GI (*Graph Isomorphism*).
Parámetros:	Dos grafos $G_1 = (V_1, E_1)$ y $G_2 = (V_2, E_2)$, del mismo orden $\mid V_1 \mid = \mid V_2 \mid = n$.
Pregunta:	¿Existe un isomorfismo $\tau : V_1 \to V_2$ tal que una arista $(u, v) \in E_1$ si y solo si $(\tau(u), \tau(v)) \in E_2$?

Una forma de tratar este problema por fuerza bruta consistiría en calcular todas las posibles premutaciones de n elementos para ver si una de ellas cumple las concidiones requeridas para el isomorfismo de grafos. Puesto que el número de permutaciones de n elementos es $n!$, esta aproximación sería claramente exponencial e intratable para valores de n mínimamente grandes.

Sin embargo, el problema puede tener casos en los cuales la solución sea relativamente sencilla ya que los isomorfismos de grafos conservan muchas de las propiedades del grafo. Por ejemplo, podemos calcular de forma muy rápida los grados de todos los nodos y dado un isomorfismo τ tenemos que el grado de u es siempre el mismo que el grado de $\tau(u)$ para cualquier nodo u.

Esto nos puede restringir el problema a las permutaciones que se puedan construir manteniendo los grados. Es algo parecido a lo que sucede con el algoritmo de la factorización de enteros, que en casos particulares puede tener una solución sencilla, pero debemos considerar el caso peor para la utilización de este problema como base de la seguridad de otros algoritmos.

El problema del isomorfismo de grafos se sabe desde hace tiempo que no es puramente exponencial y que existen cotas subexponenciales, diferentes para algunos casos particulares, pero ciertamente mejores que la exponencial.

Muy recientemente se han producido novedades en relación con este problema. Concretamente en el año 2015 el profesor László Babai presentó un algoritmo quasi-polinómico para el problema del isomorfismo de grafos. Esta publicación se puede consultar por ejemplo en [1]. Sin embargo el 4 de enero de 2017, el propio autor se retractaba de su resultado al haber sido detectado un error en la demostración. Poco después, el 9 de enero de 2017, anunciaba una versión corregida del problema que parece que se ha confirmado como correcta.

La complejidad que presenta este algoritmo es $e^{(\ln n)^{O(1)}}$. Las consecuencias que esto puede tener en la utilización del problema del isomorfismo de grafos en algoritmos de seguridad criptográfica es algo que escapa a nuestros conocimientos, aunque no parece que se haya producido ninguna reacción importante en relación con este resultado. En cualquier caso, conviene estar atentos a los avances en este sentido por si debemos usar técnicas alternativas.

La historia de este descubrimiento se puede consultar en [8].

3.3 EL PROBLEMA DEL CICLO HAMILTONIANO

Nombre:	Problema HC (*Hamiltonian Cycle*).
Parámetros:	Un grafo $G = (V, E)$.
Pregunta:	¿Existe un ciclo hamiltoniano en G?

De nuevo nos encontramos con un problema que podemos resolver considerando todas las permutaciones de los nodos, ya que si existe una de ellas que nos proporcione un ciclo hamiltoniano, habríamos resuelto el problema. Pero este algoritmo es claramente ineficiente puesto que requiere un análisis de $n!$ permutaciones de los n nodos.

Una forma algo más inteligente sería partir de un nodo cualquiera y crear una rama con cada uno de los nodos adyacentes. Para cada uno de ellos, tratamos de expandir de nuevo con todos los nodos adyacentes siempre que no repitamos el nodo. De esta forma podríamos recorrer todas las opciones, pero volvemos a encontrar un número de casos exponencial.

Este problema, sin embargo, está en una situación muy distinta del problema anterior, de hecho, el problema del ciclo hamiltoniano es un problema **NPC** por lo que si existiera un algoritmo polinómico para resolverlo, tendríamos que $\mathbf{P} = \mathbf{NP}$ y se habría resuelto uno de los problemas del milenio.

La demostración de que este problema es **NPC** se puede ver en [21, Section 34.5.3]. El valor que nos sirve de certificado o testigo es precisamente el ciclo hamiltoniano, puesto que comprobar que un ciclo concreto cumple las condiciones de ciclo hamiltoniano es trivial desde el punto de vista computacional.

3.4 EL PROBLEMA DE LA 3-COLORACIÓN

Nombre:	Problema G3C.
Parámetros:	Un grafo $G = (V, E)$.
Pregunta:	¿Existe una función $\phi : V \to \{1, 2, 3\}$ tal que $\forall (u, v) \in E$, se cumple $\phi(u) \neq \phi(v)$?

Determinar si un grafo puede tener una coloración dada por dos colores es computacionalmente muy sencillo, puesto que no tenemos mas que dar un color cualquiera al primer nodo, por ejemplo 1, y todos los que estén conectados con él le asignaremos el color 2. Ahora tomamos todos los nodos con el color 2 y para cada uno de los nodos adyacentes asignamos el color 1. Si el grafo se puede colorear con dos colores, este procedimiento nos irá coloreando cada trozo del grafo (en caso de que haya zonas desconectadas, cada vez que terminemos de colorear una zona, tendremos que partir aleatoriamente de nuevo de uno de los nodos de una nueva zona para extender la coloración). Si el grafo no se puede colorear con dos colores llegaremos necesariamente a un nodo coloreado con un color que

está unido con otro del mismo color, en cuyo caso podemos responder que el grafo necesita más de dos colores para obtener una coloración.

Dado un grafo, este algoritmo nos responde de forma eficiente al problema de si se puede colorear con dos colores, o bien devolviendo una coloración, o bien diciendo que dicha coloración es imposible.

El problema es totalmente diferente en el caso de que dispongamos de tres colores. Dado un nodo cualquiera con un color asignado, por ejemplo 1, nos da dos opciones para la coloración de los nodos adyacentes, o bien 2 o bien 3. Podemos tratar de probar todas las opciones, con lo que obtendríamos un gran número de posibilidades cuando tratamos con un caso genérico (por supuesto casos particulares pueden tener soluciones sencillas).

De hecho es conocido que este problema también pertenece a la clase **NPC**. Una coloración concreta usando 3 colores serviría de certificado de que la respuesta puede ser afirmativa puesto que commprobar que todas las aristas tienen extremos con colores diferentes es sencillo computacionalmente, sin embargo, encontrar la coloración requiere algoritmos exponenciales.

Una demostración de que el problema de la 3-coloración es **NPC** se puede encontrar en [21, Problem 34-3].

4

PROBLEMAS BASADOS EN TEORÍA DE NÚMEROS

En este capítulo vamos a hacer un repaso de las cuestiones básicas de aritmética entera y modular en las que se basan los problemas utilizados para las pruebas de conocimiento cero que estudiaremos en los siguientes capítulos.

Aunque para muchos de estos problemas se han desarrollado técnicas muy avanzadas, como por ejemplo para la factorización de enteros o para el problema del logaritmo discreto, son en general problemas sencillos de plantear desde el punto de vista matemático y resolubles de forma eficiente cuando se dispone de información adicional, es decir, del secreto.

Llamar a estos problemas *Teoría de Números* podría considerarse excesivo, podríamos simplemente llamarlos problemas artiméticos, pero hemos preferido la terminología de *Teoría de Números* para ser coherentes con muchos textos que incluyen estos métodos dentro del epígrafe de *Introducción a la Teoría de Números*.

4.1 ARITMÉTICA ENTERA Y MODULAR

Vamos a recordar los conceptos más básicos de la teoría de grupos, empezando con la definición de operación binaria:

Definición 4.1. Una *operación binaria* en un conjunto A es una aplicación $\circ : A \times A \to A$.

Según una operación binaria cumpla ciertas propiedades, diremos que:

- La operación es *asociativa* si $a \circ (b \circ c) = (a \circ b) \circ c \quad \forall a, b, c \in A$.
- La operación es *conmutativa* si $a \circ b = b \circ a \quad \forall a, b \in A$.
- El elemento $e \in A$ es un *elemento neutro* para $\circ$ si $a \circ e = e \circ a = a \quad \forall a \in A$.
- Cuando existe el neutro e, el elemento b es el *inverso* de a si $a \circ b = b \circ a = e$.

Podemos ahora dar formalmente la definición de grupo:

Definición 4.2. Un *grupo* es un conjunto G con una operación asociativa, con elemento neutro y con inverso para cada elemento. Si la operación es además conmutativa, se dice que G es un *grupo abeliano*.

Existen dos notaciones fundamentales para la representación de los grupos, que son la notación aditiva (la operación de grupo se denota $+$, el elemento neutro se denota como 0 y el inverso de a se denota $-a$) y la notación multiplicativa (la operación de grupo se denota $\cdot$, el elemento neutro se denota como 1 y el inverso de a se denota a^{-1}). Es habitual utilizar la notación aditiva para grupos conmutativos y la multiplicativa para los no conmutativos. Todos los grupos que vamos a tratar en los problemas relacionados con la Teoría de Números serán conmutativos, pero utilizaremos la notación aditiva o multiplicativa según resulte más natural una u otra.

Con la operación suma $+$ habitual, los conjuntos $\mathbb{Z}$, $\mathbb{Q}$ y $\mathbb{R}$ son grupos abelianos aditivos.

Definición 4.3. Sea G un grupo conmutativo finito y $g \in G$. Llamaremos orden de g y lo denotaremos $\mathrm{ord}(g)$ al menor entero t positivo tal que $g^t = 1$.

Proposición 4.4. *Sea* G *un grupo conmutativo finito con* n *elementos, entonces para todo* $g \in G$ *se tiene que* $g^n = 1$.

Demostración. Sean $g_1, g_2, \cdots, g_n$ los elementos de G. Está claro que si $g_i \neq g_j$ entonces $gg_i \neq gg_j$ puesto que si fueran iguales tendríamos que $g_i = g^{-1}gg_i = g^{-1}gg_j = g_j$, lo cual es una contradicción. Esto nos garantiza que el conjunto $\{gg_1, gg_2, \cdots, gg_n\}$ tiene que tener n elementos y por lo tanto ser todo G, es decir, que como conjuntos tenemos que $\{g_1, g_2, \cdots, g_n\} = \{gg_1, gg_2, \cdots, gg_n\}$ puesto que solo hay una reordenación. Como el producto es conmutativo, si multiplicamos todos los elementos tenemos que $g_1g_2\cdots g_n = g^n g_1 g_2 \cdots g_n$ y multiplicando por el inverso de $g_1g_2\cdots g_n$ obtenemos el resultado que buscamos. □

Este resultado es también cierto para grupos no conmutativos, pero la demostración no es tan elemental como en este caso, que es el que vamos a necesitar en esta memoria, por eso hemos hecho una demostración directa. Esta proposición nos garantiza que el orden de un elemento en un grupo finito tiene que ser menor o igual que el número de elementos del grupo.

Definición 4.5. Un grupo abeliano finito G diremos que es cíclico cuando exista algún elemento g tal que $\mathrm{ord}(g) = |G|$. En caso de existir, un elemento g que cumpla dichas propiedades se llamará un generador del grupo G.

Definición 4.6. Un *anillo* es un conjunto A con dos operaciones, $+$ y $\cdot$ (suma y producto), tales que:

- $(A, +)$ es un grupo abeliano con elemento neutro 0.
- El producto es asociativo y tiene elemento neutro que se denota por 1. Este elemento consideraremos que es distinto del neutro aditivo ($1 \neq 0$).
- El producto es distributivo con respecto a la suma, es decir, $\forall a, b, c \in A$ se tiene $a \cdot (b + c) = a \cdot b + a \cdot c$ y $(b + c) \cdot a = b \cdot a + c \cdot a$.

El anillo es *conmutativo* si lo es el producto. En general, no todos los elementos de un anillo son invertibles (es decir, tienen inverso para la multiplicación), por ejemplo el elemento 0 nunca lo es. Un anillo conmutativo donde todo elemento distinto de cero es invertible, se dice que es un *cuerpo*.

Definición 4.7. Sea A un anillo conmutativo, A^* denotará el *conjunto de los elementos invertibles en A*. Así, con la multiplicación usual $\mathbb{Z}^* = \{-1, 1\}$.

Los elementos invertibles también los llamaremos *unidades* del anillo A.

Si A es un cuerpo, por definición $A^* = A \setminus \{0\}$, por ejemplo $\mathbb{Q}^* = \mathbb{Q} \setminus \{0\}$ ó $\mathbb{R}^* = \mathbb{R} \setminus \{0\}$.

Proposición 4.8. *El conjunto de las unidades de un anillo conmutativo* A *forma un grupo conmutativo con la multiplicación.*

Demostración. La multiplicación de dos unidades $a, b \in A^*$ es una unidad puesto que si c es inverso de a y d inverso de b, $(ab)(dc) = a(bd)c = a \cdot 1 \cdot c = ac = 1$. La operación de multiplicación es asociativa y conmutativa por serlo en A, el 1 está en A y todo elemento tiene inverso por definición de A^*. □

Definición 4.9. Sean A y B dos anillos conmutativos y $f : A \to B$ una aplicación. Diremos que es un homomorfismo de anillos si $f(a_1 + a_2) = f(a_1) + f(a_2)$, $f(a_1 a_2) = f(a_1)f(a_2)$, $f(0) = 0$ y $f(1) = 1$. Si la aplicación f es biyectiva diremos que f es un isomorfismo y que los anillos A y B son isomorfos.

Proposición 4.10. *Sea $f : A \to B$ un isomorfismo de anillos conmutativos. Entonces para todo elemento invertible a de A, $f(a)$ es un elemento invertible de B, la aplicación f restringida a los conjuntos A^* y B^* es un isomorfismo de grupos abelianos.*

Demostración. Si a es invertible con inverso c tenemos que $1 = f(1) = f(ac) = f(a)f(c)$ por lo que $f(c)$ es inverso de $f(a)$ y por lo tanto $f(a) \in B^*$. Por otro lado, si b es un elemento invertible de B con inverso d, al ser f biyectiva podemos encontrar unos valores a y c únicos tales que $f(a) = b$ y $f(c) = d$, por lo tanto $f(ac) = f(a)f(c) = bd = 1 = f(1)$, pero al ser f biyectiva deducimos que $ab = 1$ y por lo tanto f es suprayectiva considerada como aplicación entre A^* y B^*. Eso prueba que la restricción de f es un isomorfismo de grupos abelianos puesto que evidentemente conserva el producto y el elemento neutro al conservarlo f. □

Definición 4.11. Sean A y B dos anillos conmutativos, denotaremos $A \times B$ al anillo cuyos elementos son los pares ordenados (a, b) con $a \in A$ y $b \in B$, junto con la suma y el producto componente a componente. El elemento neutro para la suma en $A \times B$ será $(0, 0)$ y para el producto $(1, 1)$.

Proposición 4.12. *Sean A y B dos anillos conmutativos, entonces $(A \times B)^*$ es el subconjunto de $A \times B$ formado por los pares $A^* \times B^*$.*

Demostración. Si a es invertible con inverso c y b es invertible con inverso d entonces $(a, b) \cdot (c, d) = (ac, bd) = (1, 1)$. Recíprocamente si (a, b) es un elemento invertible de $A \times B$ tenemos un elemento (c, d) tal que $(a, b) \cdot (c, d) = (1, 1)$, pero eso implica que $ac = 1$ y $bd = 1$ por lo que $a \in A^*$ y $b \in B^*$. □

Definición 4.13. Un entero d se dice un *divisor* de un entero a si existe un entero c tal que $dc = a$. También se dice que d divide a a o que a es un múltiplo de d.

Definición 4.14. Dados dos enteros a y b y otro entero $n > 1$, diremos que a y b son **congruentes módulo** n si $a - b$ es un múltiplo de n. Esta relación se denotará por $a \equiv b \pmod{n}$ o simplemente $a \equiv b \bmod n$.

La relación de congruencia módulo n es una relación de equivalencia que divide al conjunto de los números enteros en n clases de equivalencia distintas. Al conjunto cociente formado por estas clases de equivalencia se denotará $\mathbb{Z}_n$.

Una posible elección de representantes para las clases de equivalencia módulo n son los valores $\{0, 1, 2, \cdots, n-1\}$. Si fijamos este conjunto de representantes denotaremos $- \bmod n : \mathbb{Z} \to \{0, 1, \cdots, n-1\}$ a la aplicación que dado un elemento $a \in \mathbb{Z}$ nos da el representante de su clase de equivalencia módulo n, lo llamaremos el reducido módulo n de a.

Es habitual utilizar los valores $\{0, 1, 2, \cdots, n-1\}$ para representar también las clases de equivalencia de $\mathbb{Z}_n$, por lo tanto podemos decir $3 \in \mathbb{Z}_7$ para referirnos a la clase de equivalencia del 3 en el conjunto cociente $\mathbb{Z}_n$. Esto en general simplifica las notaciones y no causa problemas de interpretación.

La utilización de $- \bmod n$ como un operador entre $\mathbb{Z}$ y $\{0, 1, \cdots, n-1\}$ no es del todo estándar, pero la utilizaremos en esta memoria porque resulta cómoda para trabajar en aritmética modular y en la práctica, casi todos los lenguajes de programación tienen el reducido módulo n como una función más.

Si $a \equiv b \pmod n$ entonces para cualquier $c \in \mathbb{Z}$ tenemos que $a + c \equiv b + c \pmod n$ (porque $(b + c) - (a + c) = b - a$ que es un múltiplo de n) y también $ac \equiv bc \pmod n$ (porque $bc - ac = (b - a)c$ que es un múltiplo de n por serlo $b - a$). Esta propiedad nos permite probar que si $a \equiv b \pmod n$ y $c \equiv d \pmod n$ entonces $a + c \equiv b + c \equiv b + d \pmod n$ y también $ac \equiv bc \equiv bd \pmod n$.

Dicho de otra forma, la relación de congruencia módulo n respeta la suma y el producto por lo que podemos dotar a $\mathbb{Z}_n$ de estructura de anillo con la estructura heredada de $\mathbb{Z}$, es decir, dadas dos clases a y b, la suma $a + b$ es la clase de la suma de cualquier representante de a y cualquier representante de b, siendo esta definición independiente del representante elegido puesto que la congruencia respeta la suma. Lo mismo sucede con el producto, el producto de las clases, es la clase del producto de dos representantes cualesquiera de las clases.

Esta definición se simplifica mucho en la práctica cuando tenemos elegidos unos representantes de las clases y un operador como $\bmod n$ puesto que podemos partir del conjunto $\{0, 1, \cdots, n-1\}$, hacer las operaciones de suma y producto con esos representante y si el resultado final se sale fuera del conjunto de representantes, aplicar el operador $\bmod n$ al resultado. De esta forma podemos decir que en el anillo $\mathbb{Z}_2$ se tiene que $1 + 1 = 0$ puesto que $1 + 1$ sería 2 pero al aplicar el operador $\bmod 2$ obtenemos el resultado 0.

Para prodecer de esta forma, es fundamental tener un método para calcular el reducido módulo n de cualquier número, pero eso nos lo va a proporcionar precisamente el algoritmo de la división.

Proposición 4.15. *Sean a, n dos números enteros con $n > 1$, $q = \lfloor \frac{a}{n} \rfloor$, entonces $a \bmod n = a - nq$. Estos valores q y $a \bmod n$ son precisamente el cociente y resto de la división entre a y n.*

Demostración. Por definición de $q = \lfloor \frac{a}{n} \rfloor$ sabemos que q es el mayor entero menor o igual que $\frac{a}{n}$, por lo tanto $q \leqslant \frac{a}{n} < q + 1$ y de ahí se deduce que $qn \leqslant a < nq + n$ por lo que $0 \leqslant a - nq < n$, es decir, que el entero $a - nq$ está en el conjunto $\{0, 1, \cdots, n-1\}$ tal y como habíamos exigido a $a \bmod n$. Además, con esta definición $a - (a \bmod n) = nq$ por lo que $a \equiv a \bmod n$. □

Definición 4.16. Dados dos enteros a y b, llamaremos máximo común divisor de dichos números (y lo denotaremos $\mathrm{mcd}(a, b)$) al mayor entero no negativo d tal que d divide al mismo tiempo a a y a b. Este número siempre existe puesto que 1 es siempre divisor de ambos, por lo que el conjunto de los divisores comunes no es nunca vacío. En el caso particular en que este sea el único divisor común diremos que a y b son coprimos.

Para el cálculo del máximo común divisor de dos números, es muy útil el siguiente resultado:

Lema 4.17. *Sean a y b dos enteros positivos. Entonces el conjunto de divisores comunes de a y b es el mismo que el de los divisores comunes de b y $a \bmod b$.*

Demostración. Por definición $a \bmod b = a - bq$ donde $q = \lfloor \frac{a}{b} \rfloor$. Si c es divisor de a y b, entonces $a = a'c$ y $b = b'c$ y por lo tanto $a - bq = c(a' - b'q)$, lo cual prueba que c es divisor común de b y $a \bmod b$.

Recíprocamente, si c es divisor común de b y $a \bmod b$ tenemos que $b = cb'$ y $a - bq = cr'$ por lo que $a = c(b'q + r')$, lo cual prueba que c es un divisor común de a y b. □

Algoritmo 4.18 (Algoritmo de Euclides)**.** Dados dos enteros a y b entonces podemos encontrar el máximo común divisor $d = \mathrm{mcd}(a, b)$ mediante el siguiente algoritmo:

1. Inicializar el vector fila $M := (a, b)$
2. Mientras que $M_2 \neq 0$ asignar $q := \lfloor \frac{M_1}{M_2} \rfloor$ y $M := M \begin{pmatrix} 0 & 1 \\ 1 & -q \end{pmatrix}$.
3. Al terminar tendremos $d := M_1$.

Demostración. Fijémonos que en el primer paso tenemos que $q := \lfloor \frac{a}{b} \rfloor$ y por lo tanto al hacer $(a, b) \begin{pmatrix} 0 & 1 \\ 1 & -q \end{pmatrix}$ obtenemos $(b, a \bmod b)$ y utilizando el lema anterior, deducimos que los divisores comunes de los dos elementos del nuevo M son los mismos que los divisores comunes de a y b. El mismo razonamiento en todos los pasos nos prueba que esta propiedad se mantiene siempre, además los valores se van reduciendo porque $a \bmod b$ es estrictamente menor que b. La sucesión tendrá que terminar en algún momento y la única posibilidad es que termine porque M_2 sea 0. En ese caso los divisores comunes de a y b son los mismos que los de M_1 y 0, pero los divisores de M_1 y 0 son los divisores de M_1 y el máximo de todos ellos es M_1 por lo que tiene que ser el máximo común divisor de a y b. □

Este algoritmo se puede extender para calcular también los coeficientes r y s que nos dan el máximo común divisor como combinación de a y b.

Algoritmo 4.19 (Algoritmo de Euclides Extendido)**.** Dados dos enteros a y b entonces podemos encontrar el máximo común divisor $d = \mathrm{mcd}(a, b)$ y valores enteros s y t tales que $d = sa + tb$ mediante el siguiente algoritmo:

1. Inicializar la matriz $M := \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ a & b \end{pmatrix}$
2. Mientras que $M_{3,2} \neq 0$ asignar $q := \lfloor \frac{M_{3,1}}{M_{3,2}} \rfloor$ y $M := M \begin{pmatrix} 0 & 1 \\ 1 & -q \end{pmatrix}$.
3. Al terminar tendremos $r := M_{1,1}$, $s := M_{2,1}$ y $d := M_{3,1}$.

Demostración. El comportamiento de la fila inferior de la matriz M es precisamente la misma que teníamos en el Algoritmo de Euclides, por lo que necesariamente la matriz M terminará teniendo $\mathrm{mcd}(a, b)$ y 0 en la fila inferior.

Para completar la demostración fijémonos en que para los distintos valores de M que aparecen en el algoritmo, el producto $(a, b, -1)\, M$ es constantemente igual a $(0, 0)$. Para darse cuenta de eso, símplemente veamos que se cumple para la primera matriz ya que $a \cdot 1 + b \cdot 0 - 1 \cdot a = 0$ y $a \cdot 0 + b \cdot 1 - 1 \cdot b = 0$. En cada paso multiplicamos por la derecha

por una matriz, pero como el producto de matrices es asociativo, la propiedad se mantiene puesto que

$$(a,b,-1)\left(M\begin{pmatrix}1&0\\0&1\\a&b\end{pmatrix}\right)=((a,b,-1)M)\begin{pmatrix}1&0\\0&1\\a&b\end{pmatrix}=(0,0)\begin{pmatrix}1&0\\0&1\\a&b\end{pmatrix}=(0,0).$$

Esto nos prueba que al terminar el algoritmo tendremos $aM_{1,1}+bM_{2,1}=M_{3,1}$ o lo que es lo mismo $ar+bs=d$. □

El algoritmo de Euclides extendido es muy útil para el cálculo de los elementos invertibles o unidades de $\mathbb{Z}_n$.

Proposición 4.20. *El conjunto de las unidades del anillo $\mathbb{Z}_n$ es precisamente el conjunto de las clases de equivalencia de a para aquellos valores a coprimos con n.*

Demostración. Si a es coprimo con n, por el algoritmo de Euclides extendido tenemos r y s tales que $ra+sn=1$ pero entonces $ra\equiv 1 \pmod n$ y por lo tanto a es una unidad con inverso r.

Recíprocamente, si a es una unidad con inverso r tenemos que $ar\equiv 1 \pmod n$ y por lo tanto $ar=1+ns$ para algún s. Si d es el máximo común divisor de n y a, en particular d divide a ar y a ns por lo que divide a $ar-ns=1$, lo cual prueba que d necesariamente ha de ser 1. □

Observación. Tal y como hemos visto en la demostración de la proposición anterior, dado a un elemento invertible de $\mathbb{Z}_n$, el inverso de a módulo n se puede calcular como el coeficiente r que hace que $ra+sn=1$.

Definición 4.21. El número de unidades de $\mathbb{Z}_n$, es decir, el número de elementos de $\mathbb{Z}_n^*$ se denota $\varphi(n)$ que se conoce como la función φ de Euler. Esta función se define también para el 1 con el valor $\varphi(1)=1$.

Proposición 4.22. *Sea p un número primo, entonces $\mathbb{Z}_p$ es un cuerpo.*

Demostración. Tenemos que demostrar que todos los elementos distintos de 0 son unidades, pero eso es claro porque si p es primo, no puede tener un factor común distinto de 1 con ninguno de los elementos $\{1,2,3,\cdots,p-1\}$ porque ninguno de ellos es múltiplo de p. □

Observación. De la proposición anterior, deducimos que si p es un número primo, entonces $\varphi(p)=p-1$.

Teorema 4.23 (de Euler)**.** *Para cualquier $a\in\mathbb{Z}_n^*$ se tiene que $a^{\varphi(n)}\equiv 1 \pmod n$.*

Demostración. Puesto que las unidades forman un grupo abeliano finito, esto es un caso particular de la Proposición 4.4. □

Proposición 4.24. *Sea G un grupo abeliano finito de n elementos y $g\in G$, entonces $\mathrm{ord}(g)$ divide a n.*

Demostración. Supongamos por reducción al absurdo que $\mathrm{ord}(g)$ no divide a n y por lo tanto tenemos $d=\mathrm{mcd}(\mathrm{ord}(g),n)$ estrictamente menor que $\mathrm{ord}(g)$. El Algoritmo de Euclides Extendido nos permite encontrar r y s tales que $d=\mathrm{ord}(g)r+ns$, por lo tanto $g^d=(g^{\mathrm{ord}(g)})^r(g^n)^s=1$, lo cual es una contradicción con que $\mathrm{ord}(g)$ sea el menor entero tal que $g^{\mathrm{ord}(g)}=1$. □

Corolario 4.25. *Para todo* $a \in \mathbb{Z}_n^*$ *se tiene que* $\mathrm{ord}(a)$ *divide a* $\varphi(n)$.

Teorema 4.26 (Teorema Chino de los Restos)**.** *Sean* m *y* n *dos números enteros coprimos, entonces tenemos un isomorfismo de anillos* $f : \mathbb{Z}_{mn} \to \mathbb{Z}_m \times \mathbb{Z}_n$ *dado por* $f(x) = (x \bmod m, x \bmod n)$.

Demostración. Los operadores $- \bmod m$ y $- \bmod n$ respetan la suma y el producto, por lo que f respeta estas dos operaciones. El punto clave de la demostración es ver cual es la función inversa de f y esta viene dada del siguiente modo: Utilizando el Algoritmo de Euclides Extendido tomemos r y s tales que $rm + sn = 1$. Dado un par de elementos $a \in \mathbb{Z}_m$ y $b \in \mathbb{Z}_n$ podemos calcular $x = asn + brm$ y vemos que $a = a1 = arm + asn = x + (a - b)rm$ por lo que $a \equiv x \bmod m$. De forma similar se ve que $b \equiv x \bmod n$ por lo que concluimos que $f(x) = (a, b)$. Formalmente esto prueba que f es suprayectiva, pero realmente es la inversa porque también es inyectiva: Supongamos que x cumple que $f(x) = (0, 0)$. Esto significa que x es un múltiplo de m y de n, es decir, $x = um = vn$ para ciertos valores u y v, pero entonces $x = x1 = x(rm + sn) = xmr + xns = vnmr + umns = (vr + us)mn$, pero esto nos dice que x es un múltiplo de mn y por lo tanto 0 en $\mathbb{Z}_{mn}$. □

Corolario 4.27. *Seam* m *y* n *enteros coprimos, entonces* $\varphi(mn) = \varphi(m)\varphi(n)$.

Demostración. El isomorfismo de anillos que nos da el Teorema Chino de los Restos nos proporciona una biyección entre los conjuntos de unidades $\mathbb{Z}_{mn}^*$ y $\mathbb{Z}_m^* \times \mathbb{Z}_n^*$, por lo que el número de elementos de estos dos conjuntos es el mismo, pero dicho número es en el primer caso $\varphi(mn)$ y en el segundo $\varphi(m)\varphi(n)$. □

Corolario 4.28. *Sean* p *y* q *dos primos distintos, entonces* $\varphi(pq) = (p - 1)(q - 1)$.

Demostración. Por ser p y q primos distintos, está claro que no tienen ningún factor común distinto de 1 y por lo tanto $\varphi(pq) = \varphi(p)\varphi(q) = (p - 1)(q - 1)$. □

Observación. Sea $n = pq$ el producto de dos primos distintos, entonces es equivalente conocer p y q o conocer n y $\varphi(n)$.

Demostración. Si conocemos p y q claramente conocemos $n = pq$ y $\varphi(n) = (p - 1)(q - 1)$ haciendo unas simples multiplicaciones. Recíprocamente, si conocemos n y $\varphi(n) = (p - 1)(q - 1) = n - (p + q) + 1$ podemos deducir el valor de $p + q = n - \varphi(n) + 1$ por lo que conocemos la suma y el producto de dos números. Resolviendo en el conjunto de los números enteros la ecuación de segundo grado $x^2 - (p + q)x + pq = 0$ podemos obtener los valores de p y q y eso se puede hacer con el cálculo de una raíz cuadrada, que es un problema polinómico en $\mathbb{Z}$. □

Definición 4.29. Sea $a \in \mathbb{Z}_n^*$. Se dice que a es un *residuo cuadrático* módulo n, o un *cuadrado* módulo n, si existe un $x \in \mathbb{Z}_n^*$ tal que $x^2 \equiv a \bmod n$. Si no existe dicho x, entonces a se llama un *no-residuo cuadrático* módulo n.

Al conjunto de todos los residuos cuadráticos módulo n de $\mathbb{Z}_n^*$ los denotaremos como Q_n. Al conjunto de los no-residuos cuadráticos lo denotamos como $\overline{Q_n}$.

Ejemplo 4.30. Si tomamos $n = 4$, los no-residuos cuadráticos son 2 y 3, y el único residuo cuadrático es 1:

$$1^2 \equiv 1 \bmod 4 \qquad 2^2 \equiv 0 \bmod 4 \qquad 3^2 \equiv 1 \bmod 4$$

Observación. Por definición $0 \notin \mathbb{Z}_n^*$, y por tanto $0 \notin Q_n$ ni $0 \notin \overline{Q_n}$.

Definición 4.31. Sea $a \in Q_n$. Si $x \in \mathbb{Z}_n^*$ satisface $x^2 \equiv a \bmod n$, entonces x se llama *raíz cuadrada* módulo n de a.

Nuestro objetivo es estudiar el problema de cuando un elemento $a \in \mathbb{Z}_n^*$ es un residuo cuadrático y en caso de serlo cómo calcular una raíz cuadrada.

Empecemos viendo que cuando n es un primo impar, el número de residuos cuadráticos y de no residuos es el mismo.

Proposición 4.32. *Sea $p > 1$ un número primo, entonces $|Q_p| = |\overline{Q_p}| = \frac{p-1}{2}$.*

Demostración. Consideremos la función $s : \mathbb{Z}_p^* \to Q_p$ dada por $s(c) = c^2$. Por definición de Q_p esta función es suprayectiva y dado un elemento $a \in Q_p$ las antiimágenes por s serán sus raíces cuadradas. Sabemos que tiene que tener al menos dos raíces, puesto que si $a^2 \equiv c \bmod p$ entonces $(-a)^2 \equiv c \bmod p$ y $a \neq -a$ puesto que estamos suponiendo $a \neq 0$ y p primo impar. Lo importante es que no es posible que haya más de dos raíces, pero eso es cierto porque las raíces cuadradas de c son las que aparecen en la factorización del polinomio $x^2 - c$ y este polinomio no puede tener más de dos raíces por ser $\mathbb{Z}_p[x]$ un dominio de factorización única al ser $\mathbb{Z}_p$ un cuerpo.

Con este razonamiento deducimos que el número de elementos de Q_p ha de ser la mitad que el de $\mathbb{Z}_p^*$ que es precisamente $\varphi(p) = p - 1$. La otra mitad de los valores serán necesariamente no residuos, es decir, elementos de $\overline{Q_p}$. □

Cuando el número de elementos no es primo, en particular cuando podemos descomponerlo como producto de dos primos impares distintos (que será el ejemplo fundamental que necesitaremos a lo largo de la memoria), el problema se puede descomponer en dos problemas, uno para cada factor. Vamos a ver el resultado en general:

Proposición 4.33. *Sea n y m dos números enteros positivos coprimos. Entonces un elemento $x \in \mathbb{Z}_{mn}$ es un residuo cuadrático si y sólo si $x \bmod m$ y $x \bmod n$ son residuos cuadráticos en $\mathbb{Z}_m$ y $\mathbb{Z}_n$ respectivamente.*

Además, si a y b son raíces cuadradas de $x \bmod m$ y $x \bmod n$ en sus correspondientes anillos, podemos combinarlas mediante el Teorema Chino de los Restos para obtener una raíz cuadrada de x en $\mathbb{Z}_{mn}$.

En particular, esto es cierto cuando queramos estudiar los residuos cuadráticos en $\mathbb{Z}_{pq}$ con p y q dos primos impares distintos, que son en particular coprimos.

Demostración. Esto es consecuencia directa del isomorfismo de anillos $f : \mathbb{Z}_{mn} \to \mathbb{Z}_m \times \mathbb{Z}_n$ dado por el Teorema Chino de los Restos, ya que si $c^2 \equiv x \bmod mn$ entonces podemos tomar $a = c \bmod m$ y $b = c \bmod n$ de forma que $a^2 \equiv c^2 \equiv x \bmod m$ y $b^2 \equiv c^2 \equiv x \bmod n$. El el sentido contrario, si tenemos a y b raíces cuadradas de $x \bmod m$ y $x \bmod n$, entonces podemos combinarlas a un $c \in \mathbb{Z}_{mn}$ tal que $x \equiv a^2 \equiv c^2 \bmod m$ y $x \equiv b^2 \equiv c^2 \bmod n$ por lo que $f(x - c^2) = 0$ y usando que f es inyectiva, tenemos que $x \equiv c^2 \bmod mn$. □

Corolario 4.34. *Sea $n = pq$ el producto de dos primos distintos, entonces el número de residuos cuadráticos módulo n es $\frac{(p-1)(q-1)}{4}$.*

Demostración. Hemos visto que un residuo cuadrático módulo n es combinación de los residuos cuadráticos de $\mathbb{Z}_p$ y $\mathbb{Z}_q$, pero hay $\frac{p-1}{2}$ formas de elegir uno en $\mathbb{Z}_p$ y $\frac{q-1}{2}$ formas de elegir otro en $\mathbb{Z}_q$, por lo que en total tendremos $\frac{(p-1)(q-1)}{4}$. □

Para identificar los residuos cuadráticos en el caso $\mathbb{Z}_p$ con p primo, disponemos de una herramienta muy útil:

Definición 4.35. Dados un primo impar p y un entero a, se define el *Símbolo de Legendre* y se denota $\left(\frac{a}{p}\right)$ como

$$\left(\frac{a}{p}\right) = \begin{cases} 0, & \text{si } a \equiv 0 \bmod p \\ 1, & \text{si } a \in Q_p \\ -1, & \text{si } a \in \overline{Q_p} \end{cases}$$

La definición de este símbolo requiere que p sea un número primo, pero se puede extender la definición para el caso general obtiendo lo que se conoce como Símbolo de Jacobi (también denominado Símbolo de Jacobi-Kronecker en algunos textos). La definición es la siguiente:

Definición 4.36. Sea n un entero impar positivo cuya descomposición en factores primos es $n = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r}$ y sea a un entero. Entonces definimos el *símbolo de Jacobi* de a y n como

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{e_1} \cdot \left(\frac{a}{p_2}\right)^{e_2} \cdot \ldots \cdot \left(\frac{a}{p_t}\right)^{e_t}$$

En la parte derecha de la igualdad, los símbolos que aparecen son los de Legendre puesto que los p_i son primos. En la parte izquierda tenemos el Símbolo de Jacobi que será igual al símbolo de Legendre cuando tengamos el caso n primo.

Notemos que con esta definición, cuando n no es primo, podemos perder la propiedad de que $\left(\frac{a}{n}\right) = 1$ nos garantice que a es un cuadrado módulo n. Un ejemplo importante de este hecho es el caso en que n sea el producto de dos primos impares p y q.

Proposición 4.37. *Sean p y q dos primos impares, $n = pq$ y $a \in \mathbb{Z}_n^*$, entonces $\left(\frac{a}{n}\right) = 1$ si y sólo si se cumple alguna de estas dos condiciones, o bien a es un residuo cuadrático módulo n, o bien $a \bmod p$ y $a \bmod q$ son ambos no residuos cuadráticos.*

Demostración. Por definición $\left(\frac{a}{n}\right) = \left(\frac{a}{p}\right)\left(\frac{a}{q}\right)$ y como $\left(\frac{a}{p}\right)\left(\frac{a}{q}\right)$ ha de ser 1, pero como estos valores sólo pueden ser 1 ó -1, tenemos que, o bien los dos valen 1, o bien los dos valen -1, que son las dos condiciones que hemos considerado. □

Observación. Una vez introducidos los Símbolos de Legendre y Jacobi, ampliaremos la notación de los conjuntos de residuos cuadráticos. Escribiremos $\mathbb{Z}_n^Q$ para el conjunto de los valores con Símbolo de Jacobi 1, para el subconjunto que son residuos cuadráticos utilizaremos $\mathbb{Z}_n^{Q+} = Q_n$, y finalmente, el subconjunto $\mathbb{Z}_n^{Q-}$ serán los no-residuos cuadráticos con Símbolo de Jacobi 1. Se sigue que $\mathbb{Z}_n^Q = \mathbb{Z}_n^{Q+} \cup \mathbb{Z}_n^{Q-}$ y $\mathbb{Z}_n^{Q+} \cap \mathbb{Z}_n^{Q-} = \emptyset$. Esta notación no es muy habitual en Teoría de Números, pero es habitual en el caso de las pruebas de concimiento cero que veremos más adelante.

A pesar de perder la capacidad de identificar los residuos cuadráticos tal como hacía el Símbolo de Legendre, la gran ventaja del Símbolo de Jacobi son las propiedades que tiene y que permiten su cálculo sin necesidad de conocer la factorización de los números involucrados.

Proposición 4.38. *Sean $a, b \in \mathbb{Z}$ y sean m, n enteros positivos impares. Entonces se cumplen las siguientes propiedades del símbolo de Jacobi:*

1. *Si* $a \equiv b \bmod n$ *entonces* $\left(\frac{a}{n}\right) = \left(\frac{b}{n}\right)$.
2. $\left(\frac{a^2}{n}\right) = 1$.
3. $\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right)$.
4. $\left(\frac{-1}{n}\right) = (-1)^{(n-1)/2}$.
5. $\left(\frac{2}{n}\right) = (-1)^{(n^2-1)/8}$.
6. $\left(\frac{m}{n}\right) = (-1)^{(n-1)(m-1)/4}\left(\frac{n}{m}\right)$.

Esta última propiedad se conoce como Ley de Reciprocidad Cuadrática.

Demostración. Ver [15, Proposition 2.13] y [15, Theorem 2.24]. □

Observación. Aunque se han utilizado aquí las fórmulas con los valores $(-1)^t$, estos valores se pueden calcular conociendo simplemente si t es par o impar, y eso se puede deducir de todas las fórmulas a partir del valor de n y m como sigue: $(n-1)/2$ es par si $n \equiv 1 \bmod 4$ e impar si $n \equiv 3 \bmod 4$. Por otro lado $(n^2-1)/8$ siempre será entero puesto que n^2-1 es el producto de dos números pares consecutivos $(n-1)(n+1)$ y uno de los dos necesariamente es un múltiplo de 4. Además, si $n \equiv \pm 1 \bmod 8$ tendremos que $(n^2-1)/8$ es par, y si $n \equiv \pm 3 \bmod 8$ entonces este número será impar. Por último, en la Ley de Reciprocidad Cuadrática, tendremos que $(n-1)(m-1)/4$ es impar si y sólo si $m \equiv n \equiv 3 \bmod 4$, lo cual se puede comprobar con un operador lógico sencillo, por ejemplo en C se podría programar como un simple `if(m & n & 2)`.

Para el cálculo efectivo de $\left(\frac{a}{b}\right)$ se procedería del siguiente modo:

Algoritmo 4.39 (Cálculo del Símbolo de Jacobi)**.** Dados dos enteros positivos a y b con b impar, esta función $\mathrm{Jacobi}(a, b)$ devuelve el símbolo de Jacobi de dichos valores de forma recursiva.

1. En caso de que a sea mayor que b, reducirlo módulo b, $a := a \bmod b$.
2. Si a es 0, devolver 0.
3. Si a es 1, devolver 1.
4. Dividir a por 2 para ponerlo en la forma $a = 2^e a'$ con a' impar. Si e es par o $b \equiv \pm 1 \bmod 8$ poner $s := 1$, en caso contrario poner $s := -1$.
5. Finalmente si $a' \equiv 3 \bmod 4$ y $b \equiv 3 \bmod 4$ devolver $-s\mathrm{Jacobi}(b, a')$ y en caso contrario devolver $s\mathrm{Jacobi}(b, a')$.

Demostración. El algoritmo simplemente usa las propiedades del Símbolo de Jacobi: primero reduce móduo b (propiedad 1), elimina casos extremos y luego quita los factores 2 calculándolos separadamente por las propiedades 2, 3 y 5. Por último da la vuelta al símbolo por la Ley de Reciprocidad Cuadrática para llamar recursivamente a la misma función con un valor del segundo parámetro estrictamente menor, lo cual garantiza que el algoritmo terminará. □

4.2 EL PROBLEMA DE LA FACTORIZACIÓN

Aunque directamente este problema no está entre los utilizados para las pruebas de conocimiento cero que vamos a estudiar en esta memoria, sí lo está de forma indirecta y por su importancia merece un apartado especial.

Nombre:	Problema de la factorización.
Parámetros:	Sea N un entero positivo no primo.
Pregunta:	Encontrar valores a y b distintos de 1 tales que $N = ab$.

Existen muchos algoritmos que nos permiten encontrar factores pequeños de N, el más evidente de todos ellos es probar directamente dichos factores, pero existen otros más sofisticados que también son muy efectivos con factores razonablemente pequeños. El caso más complicado es cuando N es producto de dos factores primos p y q de gran tamaño, $N = pq$. Ese es precisamente el caso que resultará de interés para las pruebas de conocimiento cero.

Si pretendemos buscar un factor simplemente haciendo divisiones y calculando los restos, lo que podemos garantizar es que el factor aparecerá antes de llegar a la raíz cuadrada de N, por lo que como máximo tendremos que hacer $\sqrt{N}$ divisiones. Esta aproximación al problema nos da una complejidad $O(\sqrt{N}) = O(e^{\frac{1}{2}\ln N})$ que es exponencial y totalmente imposible en el rango de valores en que nos movemos que son números de cientos e incluso miles de cifras.

Si nos centramos pues en los algoritmos que pueden resolver los casos del tipo $N = pq$ con p y q dos primos aproximadamente del mismo tamaño, lo primero que tenemos que hacer es notar que hay un caso particular que permite factorizar N cuando p y q son *demasiado* cercanos. Es el conocido como método de factorización de Fermat.

Consiste en calcular $x_i = \lceil\sqrt{N}\rceil + i$ para valores de $i = 0, 1, 2, \cdots$ y para cada uno de ellos calcular $x_i^2 - N$ que debe ser un valor relativamente pequeño. Si para alguno de estos valores llegamos a un número que es un cuadrado perfecto, escribiremos $x_i^2 - N = y_i^2$ y por lo tanto $N = x_i^2 - y_i^2 = (x_i + y_i)(x_i - y_i)$, lo que nos da una factorización de N.

Este método es muy efectivo si la diferencia entre p y q es muy pequeña, pero no es efectivo en general.

Sin embargo, nos permite introducir la técnica básica en la que se basan casi todos los métodos de factorización, que es buscar valores x e y tales que $x^2 \equiv y^2 \bmod N$, con lo que tendríamos $(x - y)(x + y) = kN$ para algún entero k. Esta relación puede dar una factorización cuando p y q no se acumulan los dos en uno de los términos $x - y$ o $x + y$. Para extraer el factor simplemente tendríamos que calcular $\mathrm{mcd}(x - y, N)$. La posibilidad de que los dos factores estén juntos en $x - y$ o en $x + y$ existe, pero los métodos de factorización lo que nos proporcionan es varias relaciones de este tipo y la probabilidad de que encontremos el factor es alta.

Para poder encontrar las relaciones del tipo $x^2 \equiv y^2 \bmod N$ se utilizan los conceptos que vamos a introducir a continuación:

Definición 4.40. Sea N un entero impar compuesto. Llamaremos una base de factores a un conjunto $\mathcal{B} = \{p_0 = -1, p_1, p_2, \cdots, p_k\}$ donde los valores p_i con $i > 0$ son números primos pequeños. Se incluye normalmente a -1 como uno de los elementos de las bases de factores aunque no es primo. Dada una base de factores $\mathcal{B}$ y un número entero x, diremos que x es

$\mathcal{B}$-smooth[1] si $x^2 \bmod N$ se puede factorizar completamente usando los factores de la base de factores $\mathcal{B}$.

Los distintos tipos de factorización intentan buscar números $\mathcal{B}$-smooth de forma que podamos conseguir una cantidad suficiente de ellos. Cada número $\mathcal{B}$-smooth nos proporciona una relación como sigue:

$$x_t^2 \equiv z_t = p_0^{e_{t0}} p_1^{e_{t1}} \cdots p_k^{e_{tk}}$$

Esta relación la almacenaremos con sus exponentes $e(z_t) = (e_{t0}, e_{t1}, \cdots, e_{tk})$ y llamaremos $d(z_t) = (d_{t0}, d_{t1}, \cdots, d_{tk}) = (e_{t0} \bmod 2, e_{t1} \bmod 2, \cdots, e_{tk} \bmod 2) \in \mathbb{Z}_2^{k+1}$ el vector que resulta de reducir todas las coordenadas e_{ij} módulo 2. Los elementos $d(z_t)$ son vectores en un espacio de dimensión $k+1$, por lo tanto, si tenemos un número suficientemente grande de ellos (con seguridad si tenemos $k+2$), estos vectores serán linealmente dependientes.

Sean $\alpha_t \in \{0, 1\}$ los coeficientes de una posible relación de dependencia lineal entre los vectores, es decir, $\sum_t \alpha_t d(z_t) = 0 \in \mathbb{Z}_2^{k+1}$. Puesto que los $d(z_t)$ eran las reducciones módulo 2 de los vectores $e(z_t)$, si aplicamos estas mismas relaciones a los vectores originales lo que obtenemos es $\sum_t \alpha_t e(z_t) = (2u_0, 2u_1, \cdots, 2u_k)$ con lo que podemos tomar $y = (-1)^{u_0} p_1^{u_1} \cdots p_k^{u_k}$ de modo que obtenemos

$$y^2 = p_0^{2u_0} p_1^{2u_1} \cdots p_k^{2u_k} = \prod_i p_i^{2u_i} = \prod_i p_i^{\sum_t \alpha_t e_{ti}} = \prod_i \prod_t \left(p_i^{e_{ti}}\right)^{\alpha_t} =$$

$$\prod_t \prod_i \left(p_i^{e_{ti}}\right)^{\alpha_t} = \prod_t \left(\prod_i p_i^{e_{ti}}\right)^{\alpha_t} = \prod_t z_t^{\alpha_t} \equiv \prod_t x_t^{2\alpha_t} \equiv \left(\prod_t x_t^{\alpha_t}\right)^2 \bmod N$$

Con lo que, tomando $x = \prod_t x_t^{\alpha_t}$ hemos obtenido una relación del tipo $y^2 \equiv x^2 \bmod N$ para cada una de las combinaciones lineales de los vectores que nos dan la dependencia lineal.

Esta técnica se ha demostrado muy exitosa y los métodos de factorización la han ido perfeccionando para encontrar números $\mathcal{B}$-smooth de forma cada vez más rápida. Los métodos que usan esta técnica son el de las fracciones contínuas (CFRAC), el de la criba cuadrática multipolinómica (MPQS) y el del la criba de los cuerpos de números (NFS).

La complejidad de estos métodos es subexponencial, concretamente la del NFS es $O(L_n[1/3, (64/9)^{1/3}$ $o(1)])$. Estos métodos no permiten factorizar números enormemente grandes (del orden de las 1000 cifras binarias), pero sí otros de moderada longitud (unos pocos cientos de cifras en binario).

La situación del problema de la factorización podría cambiar dramáticamente con la aparición de los ordenadores cuánticos puesto que existen algoritmos eficientes para la factorización con esta nueva tecnología. Sin embargo, estos ordenadores están todavía en una fase experimental.

4.3 EL PROBLEMA DE LOS RESIDUOS CUADRÁTICOS

Podemos introducir ahora el problema de decisión QR, donde dado un módulo N, compuesto e impar, decidir si un entero x con símbolo de Jacobi 1 respecto a N, es o no un residuo cuadrático:

1 Se ha preferido la terminología en inglés porque la traducción que podría ser $\mathcal{B}$-suave no se suele usar en los libros.

Nombre:	Problema de los residuos cuadráticos (QR).
Parámetros:	N un entero impar tal que $N = pq$ para p y q primos, y el entero x tal que $\left(\frac{x}{N}\right) = 1$.
Pregunta:	¿Es x un residuo cuadrático en $\mathbb{Z}_N$?

Cuando un número x es un residuo cuadrático módulo N, en particular lo es para p y para q por lo que el Símbolo de Jacobi va a ser 1. La cantidad de valores para los que esto sucede es $\frac{(p-1)(q-1)}{4}$ tal y como vimos en el Corolario 4.34. También podemos obtener que el Símbolo de Jacobi sea 1 sin ser un residuo cuadrático, eso sucede cuando x no es residuo cuadrático módulo p ni módulo q (Proposición 4.37). El número de valores para los que esto sucede es el mismo $\frac{(p-1)(q-1)}{4}$, por lo tanto, dado un número x cuyo símbolo de Jacobi sea 1, tenemos una probabilidad de $\frac{1}{2}$ de que x sea efectivamente un residuo cuadrático en $\mathbb{Z}_N$, la misma que la de que no lo sea.

Este problema es polinómicamente reducible (ver 2.14) al problema de la factorización.

Proposición 4.41. QR $\leqslant_P$ FACTORIZACIÓN

Demostración. Si conocieramos la descomposición en primos de $N = pq$, el algoritmo 4.39 aplicado a cada uno de los factores, nos resuelve el problema de calcular el símbolo de Legendre de x respecto de p y de q (que en este caso es el mismo que el Símbolo de Jacobi) por lo que podemos responder `Falso` si alguno de estos símbolos es -1 y `Verdadero` si valen 1. □

Si se desconoce la factorización de N, no se conoce a día de hoy ningún algoritmo eficiente para resolver el problema QR aparte del de intentar adivinar la respuesta, lo cual en nuestro caso se podría hacer con probabilidad 1/2.

Un problema asociado al de saber si un número es un residuo cuadrático es el de encontrar efectivamente una raíz cuadrada de dicho número. En el caso de que el módulo sobre el que trabajemos sea primo, este problema se puede resolver de forma polinómica mediante el Algoritmo de Tonelli, ver [15, Proposition 2.16]. En este algoritmo se calculan varias potencias modulares, eso se puede hacer de forma polinómica mediante el algoritmo de exponenciación modular para el cual existen distintas alternativas, ver [15, Section 2.7.2].

Utilizando el Algoritmo de Tonelli podemos resolver el problema de calcular la raíz cuadrada modular en el caso primo, y si conocemos la factorización de $N = pq$, podemos combinar las raíces cuadradas modulares para obtener una raíz cuadrada módulo N tal y como vimos en la Proposición 4.33.

Sin embargo, conocer un método para calcular raíces cuadradas modulares es un problema equivalente a la factorización a partir de un teorema debido a Rabin y que esencialmente nos dice lo siguiente: Supongamos que disponemos de un método para calcular raíces cuadradas modulares en $\mathbb{Z}_N$ con $N = pq$ producto de dos primos impares. Entonces elegimos aleatoriamente elementos $x \in \mathbb{Z}_N$ y los elevamos al cuadrado. Al resultado le aplicamos nuestro oráculo y obtenemos una raíz cuadrada y tal que $x^2 \equiv y^2 \mod N$. Esta es precisamente la relación que nos proporcionaba la factorización de N utilizando las bases de factores, así que podemos proceder del mismo modo y factorizar N con $\mathrm{mcd}(x-y, N)$. Como el oráculo lo podemos utilizar múltiples veces, si obtenemos valores para los cuales

$mcd(x - y, N) = 1$, podemos calcular nuevos candidatos hasta que obtengamos la factorización.

4.4 EL PROBLEMA DEL LOGARITMO DISCRETO

El problema del logaritmo discreto vamos a considerarlo en dos versiones diferentes a lo largo de esta memoria. La primera versión es en un grupo cualquiera:

Nombre:	Problema DL (*Discrete Logarithm*).
Parámetros:	Un grupo cíclico G de orden q primo, donde se supone difícil el problema del logaritmo discreto, un generador g, $G = \langle g \rangle$, y un elemento $y \in G$.
Pregunta:	¿Conoce P el entero $s \in \mathbb{Z}_q$ tal que $g^s = y$, o equivalentemente, $\log_g y = s$?

El grupo multiplicativo de los anillos de restos módulo p para p primo, es decir, $\mathbb{Z}_p^*$ es un conocido grupo cíclico que se usa frecuentemente:

Nombre:	Problema DL (*Discrete Logarithm*).
Parámetros:	Un número primo p un generador g de $\mathbb{Z}_p^*$, y un elemento $y \in \mathbb{Z}_p^*$.
Pregunta:	¿Conoce P el entero s tal que $g^s = y$, o equivalentemente, $\log_g y = s$?

La realidad es que la estructura del grupo G es importante en los métodos que se pueden aplicar para la resolución del problema. En la segunda versión del algoritmo podemos hacer uso no solo de la estructura multiplicativa del anillo $\mathbb{Z}_p$ sino también de la estructura de anillo, lo cual hace que existan algoritmos más efectivos, concretamente existe un algoritmo denominado Index Calculus que utilizando una técnica parecida a la de las bases de factores, permite resolver el problema en tiempo subexponencial.

Sin embargo, en grupos sin esa estructura adicional tendríamos que recurrir a algoritmos menos efectivos, como por ejemplo el algoritmo Baby-Step Giant-Step que tiene una complejidad exponencial del orden de la raíz cuadrada de n con algunos factores adicionales logarítmicos.

A la vista de esta situación, en el caso de utilizar grupos del tipo $\mathbb{Z}_p^*$ será necesario aumentar el tamaño del grupo para evitar el efecto de los algoritmos como el Index Calculus.

5 PRUEBAS DE CONOCIMIENTO CERO

Las pruebas de conocimiento cero, con siglas ZKP del inglés *Zero-Knowledge Proofs*, permiten demostrar la veracidad de una declaración, sin revelar nada más de ella. En las ZKP intervienen dos partes, el *Prover* y el *Verifier*, o Probador y Verificador. El Probador asegura que una declaración es cierta, y el Verificador quiere convencerse de ello a través de una interacción con el Probador, de modo que al final de la misma, o bien acaba convencido de que la declaración es cierta, o bien descubre, con una alta probabilidad, que el Probador mentía.

Las pruebas de conocimiento cero surgen a partir de los sistemas de pruebas interactivas, que forman una parte importante de la teoría de complejidad computacional, a las que añadiendo la propiedad de *conocimiento cero* obtenemos el subconjunto de sistemas interactivos que conforman las pruebas de conocimiento cero.

Las referencias utilizadas para este capítulo se pueden encontrar en [3, 11, 14-16, 19, 20].

5.1 SISTEMAS DE PRUEBAS INTERACTIVAS

Un *sistema de prueba interactivo* es un concepto perteneciente a la teoría computacional, que modela el intercambio de un número finito de mensajes entre dos partes, el probador P y el verificador V, con el objetivo de que P demuestre a V que una instancia de un cierto problema de decisión es $\mathtt{Verdadera}$. V es una máquina con una capacidad de cómputo limitada, a lo sumo probabilístico de tiempo de polinomial. P es computacionalmente *todopoderoso*. Al final del intercambio de mensajes, o bien V acepta que la instancia es $\mathtt{Verdadera}$, o bien la rechaza por ser $\mathtt{Falsa}$.

Definición 5.1. Se dice que un problema de decisión Q, no necesariamente en **NP**, tiene un *sistema de prueba interactivo* si tiene un protocolo de interacción polinomialmente acotado en número de mensajes que cumple:

- *Completitud* Para toda instancia q $\mathtt{Verdadera}$, del problema Q, V acepta q como $\mathtt{Verdadera}$.
- *Robustez* Para cada instancia q $\mathtt{Falsa}$, V rechaza la prueba de q con una probabilidad no menor que $\epsilon = 1 - n^{-c}$, para cualquier constante $c > 0$ y donde n es el tamaño de la instancia.

En resumen, si la instancia del problema Q que P quiere demostrar es $\mathtt{Verdadera}$, el protocolo siempre funciona, no hay falsos negativos, pero si la instancia es $\mathtt{Falsa}$, hay una pequeña probabilidad de que V la acepte como $\mathtt{Verdadera}$, pueden haber falsos positivos una probabilidad casi despreciable.

Un P o un V que no siguen el protocolo e intentan romper estas propiedades, los llamaremos un P* o V* *tramposos*.

Definición 5.2. Denominamos clase de problemas **IP** (Interactivos en tiempo Polinomial) al conjunto de problemas de decisión para los que existe un sistema de prueba interactivo.

Proposición 5.3. *NP* $\subset$ *IP*.

Demostración. Sea Q un problema **NP**. Definimos el siguiente protocolo:

1. P resuelve la instancia del problema gracias a su capacidad de cómputo ilimitada y genera el certificado para V, que existe para cualquier instancia `Verdadera` por $Q \in$ **NP** (2.12).
2. V recibe y puede verificar el certificado en tiempo polinomial. Si es válido, V acepta como `Verdadera` la instancia. Si no, rechaza la prueba.

El protocolo es completo y robusto, con probabilidad nula de falso positivo, pues si la instancia es `Falsa`, ningún P puede generar un certificado que no existe.

□

5.1.1 *Prueba Interactiva para el Problema QR*

Vamos a ver una prueba interactiva para demostrar el problema QR (Sección 4.3), donde hemos de determinar si un entero x con Símbolo de Jacobi 1 respecto a n, $x \in \mathbb{Z}_n^Q$, es un residuo cuadrático, $x \in \mathbb{Z}_n^{Q+}$.

Algoritmo 5.4 (Prueba interactiva para QR)**.**
Datos comunes: Una instancia (x, N) del Problema QR. n es el tamaño de la instancia.
Protocolo: Sea $t(n)$ un polinomio en n. P y V repiten $t(n)$ veces los siguientes pasos.

1. $P \to V: \quad u \in_R \mathbb{Z}_N^{Q+}$ (P elige aleatoriamente u en $\mathbb{Z}_N^{Q+}$).
2. $V \to P: \quad b \in_R \{0, 1\}$.
3. $P \to V:$ w, una raíz cuadrada módulo N aleatoria, de u si $b = 0$, o bien de $x \cdot u$ si $b = 1$.
4. V comprueba si:

$$w^2 \stackrel{?}{\equiv} \begin{cases} u \bmod N, & \text{si } b = 0 \\ xu \bmod N, & \text{si } b = 1. \end{cases}$$

   Si la comparación falla, V termina en rechazo. En caso contrario, vuelve al paso 1.

Tras $t(n)$ rondas, V termina y acepta la instancia como `Verdadera`, x es un residuo cuadrático módulo N.

Teorema 5.5. *El problema QR tiene un sistema de prueba interactiva.*

Demostración. El protocolo 5.4 se ejecuta $t(n)$ veces, un número de iteraciones polinomialmente asociado al tamaño n de la entrada, por lo que hay un número finito de mensajes que V, computacionalmente limitado, puede llevar a cabo.

Queda ver que el protocolo anterior es completo y robusto.

La prueba es *completa*, pues para cualquier instancia $\mathtt{Verdadera}$ de QR, $x \in \mathbb{Z}_N^{Q+}$, V acepta la prueba de P. En cada iteración, como P es computacionalmente todopoderoso, puede calcular w, una raíz cuadrada módulo N de u o xu, según el valor de b, ambos en $\mathbb{Z}_N^{Q+}$.

Para una instancia $\mathtt{Falsa}$, $x \in \mathbb{Z}_N^{Q-}$, cuando V envíe $b = 1$, si P sigue el protocolo u será un residuo cuadrático, pero $x \cdot u$ es un no-residuo cuadrático módulo N, por lo que no podrá calcular w por mucho poder computacional ilimitado que tenga. Un P tramposo podría intentar engañar a V en el caso $b = 1$ eligiendo u tal que xu es un residuo cuadrático, pero entonces u es un no-residuo cuadrático y fallaría la prueba si $b = 0$.

Una vez P se compromete con un u, residuo cuadrático o no, la probabilidad de que V lo rechace en esa iteración es $1/2$, según elija $b = 0$ ó 1. Como el protocolo se ejecuta $t(n)$ veces, la probabilidad de que un P tramposo pueda engañara V en todos es de $2^{-t(n)}$. Vemos entonces que el protocolo cumple la propiedad de *robustez*. □

5.2 PRUEBAS DE CONOCIMIENTO CERO

Entre las pruebas interactivas existe un subconjunto que llamamos de *conocimiento cero* si durante el protocolo no se puede inferir información de P, aparte de la veracidad de la instancia. En particular, aún tras realizar la prueba, y estar V convencido, éste no podría repetirla a otro verificador tomando el lugar de P.

Antes de ver una definición más formal de una prueba de conocimiento cero, necesitamos algunas definiciones previas. Seguiremos en esta sección el guión de [16] que nos pareció el más indicado.

Definición 5.6. Llamamos *Vista* a la transcripción de los mensajes intercambiados entre P y V durante la ejecución de una prueba interactiva.

Pensemos en un protocolo con 3 mensajes por iteración. En la i-ésima ronda, P envía a V el valor A_i como *compromiso*, V responde con el *reto* aleatorio B_i, y P termina enviando la *prueba* C_i. La tupla (A_i, B_i, C_i) son variables aleatorias de los posibles valores que se pueden intercambiar en una iteración. La Vista de la prueba interactiva sería la secuencia de los $t(n)$ mensajes intercambiados entre P y V:

$$(A_1, B_1, C_1, A_2, B_2, C_2, \ldots, A_{t(n)}, B_{t(n)}, C_{t(n)})$$

Una Vista sólo es de interés para una instancia $\mathtt{Verdadera}$, donde P realmente conoce si es $\mathtt{Verdadera}$. Para una instancia cuya prueba falla, o bien es una instancia $\mathtt{Falsa}$ o P no puede probarla, por ser tramposo.

Definición 5.7. Llamamos ensamble probabilístico (*ensemble* en inglés) a una familia de variables aleatorias $\{X_i\}_{i \in I}$, con I numerable.

Podemos estudiar entonces una Vista como un ensamble probabilístico. Dos Vistas serán iguales cuando las distribuciones de sus variables aleatorias sean idénticas.

Cuando tratamos con V^*, un verificador tramposo, éste podría no generar los retos B_i anteriores de manera independiente, podría incluso utilizar información previa de otras Vistas para generar los B_i en un intento de obtener información extra de P. A esta información previa la llamaremos h (historial).

Para una instancia q y un verificador cualquiera V^*, escribimos la Vista de una prueba interactiva como:

$$Vista_{P,V^*}(q,h) = (q, h, A_1, B_1, C_1, \ldots, A_{t(n)}, B_{t(n)}, C_{t(n)}).$$

El verificador tramposo V^* generará los retos B_i con un algoritmo probabilístico de tiempo polinomial F tal que

$$B_i = F(q, h, A_1, B_1, C_1, \ldots, A_{i-1}, B_{i-1}, C_{i-1}, A_i).$$

Para un verificador honesto, F se puede considerar como un generador de números aleatorios que no utiliza ninguno de los parámetros de entrada.

Definición 5.8. Un Simulador $S_{V^*}(q,h)$ es un algoritmo probabilístico de tiempo polinomial, que utiliza toda la información que V^* tiene disponible (el historial h y la función F), para generar una transcripción de una prueba interactiva, para una instancia q del problema Q, sin necesidad de interactuar con P.

Un Simulador se puede ver como un generador de ensambles probabilísticos, la Vista de una prueba. La diferencia es que una transcripción de la Vista se genera a partir de dos máquinas, P y V, a diferencia del Simulador, que se ejecuta en una única máquina.

Podemos describir, por fin, la tercera propiedad de las pruebas de conocimiento cero, que acompaña a la *completitud* y la *robustez*.

Definición 5.9 (Propiedad de conocimiento cero)**.** Un sistema de prueba interactiva (completo y robusto), para un problema de decisión Q, es de *conocimiento cero* si el ensamble $Vista_{P,V}(q,h)$ es idéntico al ensamble generado por un Simulador $S_{V^*}(q,h)$, para cualquier instancia $Verdadera$ $q \in Q$ y cualquier historial h.

A las pruebas de conocimiento cero que cumplen la definición anterior también se les llama de *conocimiento cero perfectas*, y veremos en futuras secciones otras definiciones menos restrictivas.

De manera informal, el hecho de existir un simulador de conversaciones entre P y V que es idéntico a las interacciones reales, y que puede ejecutar cualquier máquina limitada computacionalmente, significa que de las transcripciones reales podemos sacar la misma información que ejecutando nosotros el simulador, y por tanto no aprenderíamos nada nuevo de ellas.

5.2.1 *Prueba de conocimiento cero para el problema QR*

Ahora podemos volver al problema QR, que ya vimos en el Teorema 5.5 y cuya prueba interactiva cumplía *completitud* y *robustez*.

Teorema 5.10. *La prueba interactiva (5.4) del problema QR es de conocimiento cero.*

Demostración. Sea (x, N) una instancia $Verdadera$ del problema QR, $\exists y \in \mathbb{Z}_N$ tal que $y^2 \equiv x \bmod N$. En la i-ésima ronda tenemos las siguientes variables aleatorias:

1. U_i, un residuo cuadrático aleatorio enviado por P en el primer mensaje, $u \in_R \mathbb{Z}_N^{Q+}$.
2. B_i, un bit aleatorio generado por V, $b \in_R \{0, 1\}$.
3. W_i, una *prueba* de P, $w \in_R \Omega_u$ o bien $w \in_R \Omega_{xu}$, según el valor de B_i, es decir, una raíz cuadrada aleatoria módulo N de u ó xu, donde Ω_u y Ω_{xu} son el conjunto de raíces cuadradas módulo N de u y xu, respectivamente.

La Vista de una prueba para un verificador V^* cualquiera es:

$$\text{Vista}_{P,V^*}(x, N, h) = (x,\ N,\ h,\ U_1,\ B_1,\ W_1, \ldots,\ U_{t(n)},\ B_{t(n)},\ W_{t(n)}).$$

Para un V honesto, todos los B_i son variables aleatorias independientes, uniformes en $\{0, 1\}$. Para un V tramposo, la función F, probabilística en tiempo polinomial, genera los valores $b_{i+1} = F(x, N, h, v_i, u_{i+1})$, cuando $V_i = v_i$. Unimos el estudio de ambos casos suponiendo, para un V honesto, F como un generador de bits aleatorio, un lanzamiento de moneda.

Ahora que tenemos toda la información accesible a V^* podemos construir un Simulador:

Simulador para el problema QR $S_{V^*}(x, N, h)$.

Datos: (x, N), una instancia *Verdadera* del problema QR; h, transcripciones de ejecuciones previas del protocolo; v_i, transcripción de la interacción actual (i rondas).

Ejecución: Repetir para $i + 1 \leqslant t(n)$:

1. Elegir $b_{i+1} \in_R \{0, 1\}$
2. Elegir $w_{i+1} \in_R \mathbb{Z}_N^*$
3. **Si** $b_{i+1} = 0$, **entonces** calcular $u_{i+1} \equiv w_{i+1}^2 \bmod N$
   Si no, $u_{i+1} \equiv w_{i+1}^2 \cdot x^{-1} \bmod N$
4. **Si** $b_{i+1} = F(x, N, h, v_i, u_{i+1})$, **entonces** añadir la tupla $(u_{i+1},\ b_{i+1},\ w_{i+1})$ a la transcripción.
   Si no, volver al paso 1.
5. $i = i + 1$

El Simulador se diferencia del protocolo 5.4 en que, en vez de elegir primero un residuo cuadrático u_{i+1}, elige los valores b_{i+1} y w_{i+1} aleatoriamente, y a partir de ellos calcula u_{i+1}. Entonces, una vez tiene el u_{i+1} necesario para la función F, calcula el bit que V^* hubiera enviado en una interacción real, y comprueba si es el mismo bit b_{i+1} elegido. Aquí es donde vemos que el Simulador es un algoritmo probabilístico en tiempo del tipo Las Vegas (si obtenemos la tupla $i + 1$, será una tupla correcta). La probabilidad de que el bit b_{i+1} sea igual que el obtenido de F es de $1/2$. En promedio, el Simulador necesitará dos rondas por cada tupla $(u_{i+1},\ b_{i+1},\ w_{i+1},\ )$, por lo que el tiempo de ejecución esperado es polinomial.

Tenemos una prueba interactiva (5.4) que genera las vistas:

$$Vista_{P,V^*}(x, N, h) = (x, N, h, U_1, B_1, W_1, \ldots, U_{t(n)}, B_{t(n)}, W_{t(n)}),$$

y un Simulador que genera las transcripciones:

$$S_{V^*}(x, N, h) = (x, N, h, U_1', B_1', W_1', \ldots, U_{t(n)}', B_{t(n)}', W_{t(n)}').$$

Para terminar la demostración, veamos por inducción en i que son iguales, es decir, cumplen la propiedad de *conocimiento cero*.

Para el caso $i = 0$ ambos ensambles son constantes, $(x, N, h) = (x, N, h)$, tenemos la misma instancia e historial.

Suponemos cierto el caso $i-1$, es decir, el ensamble de la Vista:

$$Vista_{P,V^*}(x, N, h) = (x, N, h, U_1, B_1, W_1, \ldots, U_{i-1}, B_{i-1}, W_{i-1}),$$

es igual al del Simulador:

$$S_{V^*}(x, N, h) = (x, N, h, U_1', B_1', W_1', \ldots, U_{i-1}', B_{i-1}', W_{i-1}').$$

Siguiendo el protocolo de la prueba interactiva, se generará la siguiente tupla de la Vista, (U_i, B_i, W_i). La variable U_i se elige al inicio aleatoriamente, por lo que es independiente. La v.a. B_i se calcula con F, por lo que depende de U_i, V_{i-1} y h. W_i depende de ambos. La probabilidad de la tupla nos queda:

$$P(U_i = u, B_i = b, W_i = w) =$$

$$P(U_i = u) \cdot P(B_i = b \mid V_{i-1} = v, U_i = u, h) \cdot P(W_i = w \mid U_i = u, B_i = b)$$

Sea $\alpha = \mid \mathbb{Z}_N^{Q^+} \mid$, entonces $P(U_i = u) = \frac{1}{\alpha}$.

Denotamos $P(B_i = b \mid V_{i-1} = v, U_i = u, h) = p_b$, que dependerá de la F utilizada.

Por último, sea $\beta = \mid \Omega_u \mid = \mid \Omega_{xu} \mid$. Entonces:

$$P(W_i = w \mid U_i = u, B_i = 0) = 1/\beta, \ \forall w \in \Omega_u$$

$$P(W_i = w \mid U_i = u, B_i = 1) = 1/\beta, \ \forall w \in \Omega_{xu}$$

En total nos queda, $P(U_i = u, B_i = b, W_i = w) = \frac{p_b}{\alpha\beta}$.

Ahora veamos la tupla generada por el Simulador, (U_i', B_i', W_i'). La v.a. U_i' se calcula a partir de B_i' y W_i'. La variable B_i' depende de U_i', V_{i-1} y h por F en el paso 4. Y la v.a. W_i' se elige de manera uniforme en $\mathbb{Z}_N^*$.

La probabilidad de la tupla es:

$$P(U_i' = u, B_i' = b, W_i' = w) =$$

$$P(W_i' = w) \cdot P(B_i' = b \mid V_{i-1} = v, U_i' = u, h) \cdot P(U_i' = u \mid W_i' = w, B_i' = b)$$

Sabemos que $\mid \mathbb{Z}_N^* \mid = \alpha \cdot \beta$, por lo que $P(W_i' = w) = \frac{1}{\alpha\beta}$.

Utilizando que $w \in \Omega_u \Leftrightarrow b = 0$, $w \in \Omega_{xu} \Leftrightarrow b = 1$ y que W_i' y B_i' son independientes, la probabilidad

$$\begin{aligned}
P(U_i' = u) &= P(U_i' = u, W_i' \in \Omega_u \cup \Omega_{xu}, B_i' \in \{0, 1\}) = \\
&= \sum_{w \in \Omega_u} P(U_i' = u, W_i' = w, B_i' = 0) + \sum_{w \in \Omega_{xu}} P(U_i' = u, W_i' = w, B_i' = 1) = \\
&= \sum_{w \in \Omega_u} P(W_i' = w) P(B_i' = 0) + \sum_{w \in \Omega_{xu}} P(W_i' = w) P(B_i' = 1) = \\
&= \beta \cdot \frac{1}{\alpha\beta} \cdot (P(B_i' = 0) + P(B_i' = 1)) = \frac{1}{\alpha}
\end{aligned}$$

indica que $U_i^{'}$ tiene la misma distribución que U_i, de modo que, al calcular $b_i = F(x, N, h, v_{i-1}, u_i)$, se tiene $P(B_i^{'} = b \mid V_{i-1} = v, U_i^{'} = u, h) = p_b$, es decir, $B_i^{'}$ tiene la misma distribución que B_i.

Por construcción, dados w y b en el simulador, u tiene un único valor posible, $u \equiv w^2 x^{-b} \bmod N$, por tanto, la probabilidad de que dada la tupla (u, b, w), $U_i^{'}$ tenga el valor u condicionado a que $B_i^{'} = b$ y que $W_i^{'} = w$, es 1:

$$P(U_i^{'} = u \mid W_i^{'} = w, B_i^{'} = b) = 1$$

En total, tenemos que $P(U_i^{'} = u, B_i^{'} = b, W_i^{'} = w) = \frac{p_b}{\alpha\beta}$.

Terminamos así la inducción en i y los ensambles de la Vista y el Simulador son idénticos. Concluimos que la prueba interactiva 5.4 del problema QR es perfecta de conocimiento cero.

□

5.2.2 *Prueba de conocimiento cero para el problema de isomorfismo de grafos*

Otro problema de decisión del que podemos dar una prueba interactiva de conocimiento cero es el de determinar si dos grafos dados son isomorfos (Sección 3.2).

Teorema 5.11. *El problema GI tiene una prueba de conocimiento cero.*

Demostración.

Primero debemos dar un protocolo interactivo entre un P y un V que cumpla completitud y robustez. Después daremos un Simulador para demostrar la propiedad de conocimiento cero.

Algoritmo 5.12 (Prueba interactiva para GI)**.**

Datos comunes: Una instancia ($G_0 = (V_0, E_0)$, $G_1 = (V_1, E_1)$) del Problema GI. $\mid V_0 \mid = \mid V_1 \mid = n$ es el tamaño de la instancia.

Protocolo: P calcula el isomorfismo τ entre G_1 y G_0, es decir, $\tau(G_1) = G_0$.

Sea $t(n)$ un polinomio en n. P y V repiten $t(n)$ veces los siguientes pasos.

1. $P \to V$: $h = \pi(G_0)$, donde $\pi \in_R Sym(V_0)$.
2. $V \to P$: $b \in_R \{0, 1\}$.
3. $P \to V$: ω, tal que

$$\omega = \begin{cases} \pi & \text{si } b = 0 \\ \pi \circ \tau & \text{si } b = 1. \end{cases}$$

4. V comprueba si:

$$h \stackrel{?}{=} \begin{cases} \omega(G_0) & \text{si } b = 0 \\ \omega(G_1) & \text{si } b = 1, \end{cases}$$

es decir, si h es isomorfo a G_b por ω.

Si la comparación falla, V termina en rechazo. En caso contrario, vuelve al paso 1.

Tras $t(n)$ rondas, V termina y acepta la instancia como $Verdadera$, G_0 y G_1 son isomorfos.

Sea (G_0, G_1) una instancia $Verdadera$. Sea cual sea el reto b, P siempre puede devolver el isomorfismo ω, pues G_0 y G_1 son ambos isomorfos a h. Tenemos que el protocolo es *completo*.

Si (G_0, G_1) es $Falsa$, $G_0 \not\simeq G_1$, un P tramposo deberá adivinar el reto b antes de calcular h, pues como éste debe ser un isomorfismo de G_0 ó G_1, no podrá serlo de ambos a la vez. La probabilidad de acertar el reto y mandar el isomorfismo correcto es de $1/2$ en cada ronda, por lo que la probabilidad de que un P tramposo engañe a V es de $2^{-t(n)}$. El protocolo es *robusto*.

Sea (G_0, G_1) una instancia $Verdadera$ del problema GI. La Vista entre P y V es el ensamble

$$Vista_{P,V^*}(G_0, G_1, h) = (G_0, G_1, h, H_1, B_1, \Phi_1, \ldots, H_{t(n)}, B_{t(n)}, \Phi_{t(n)}),$$

donde la variable aleatoria H_i representa el grafo isomorfo h_i de la i-ésima ronda, la v.a. B_i el reto b_i de V a P, y Φ_i es el isomorfismo que envía P como respuesta al final de la ronda. El historial de anteriores transcripciones se representa con h.

Como en el problema QR, V^* podría utilizar un algoritmo probabilístico F, de tiempo polinomial, al calcular los retos b_i, para intentar obtener información de P. F utilizará toda la información accesible a V^* en el momento de enviar el reto.

Construimos ahora el Simulador para la prueba interactiva:

Simulador para el problema GI $S_{V^*}(G_0, G_1, h)$.

Datos: (G_0, G_1), una instancia $Verdadera$ del problema GI; h, transcripciones de ejecuciones previas del protocolo; v_i, transcripción de la interacción actual (i rondas).

Ejecución: Repetir para $i + 1 \leqslant t(n)$:

1. Elegir $b_{i+1} \in_R \{0, 1\}$
2. Elegir $\pi_{i+1} \in_R Sym(V_{b_{i+1}})$ y calcular $h_{i+1} = \pi_{i+1}(G_{b_{i+1}})$.
3. **Si** $b_{i+1} = F(G_0, G_1, h, v_i, h_{i+1})$, **entonces** añadir la tupla $(h_{i+1}, b_{i+1}, \pi_{i+1})$ a la transcripción.
   Si no, volver al paso 1.
4. $i = i + 1$

La probabilidad de que el bit b_{i+1} elegido coincida con el de la función F es $1/2$, por lo que en promedio se necesitarán dos rondas por tupla. El Simulador es un algoritmo probabilístico que se ejecuta en un tiempo estimado polinomial.

Veamos ahora que el ensamble de la $Vista_{P,V^*}(G_0, G_1, h)$, es igual al del Simulador, $S_{V^*}(G_0, G_1, h)$. Procedemos por inducción sobre i, el número de rondas.

Para $i = 0$, ambos ensambles son constantes,

$$Vista_{P,V^*}(G_0, G_1, h) = S_{V^*}(G_0, G_1, h) = (G_0, G_1, h),$$

por lo que sus distribuciones de probabilidad son idénticas y los ensambles coinciden.

Suponemos cierto para $i-1$ rondas, $P(Vista_{P,V^*} = v_{i-1}) = P(S_{V^*} = v_{i-1})$.

Siguiendo el protocolo, se generarán las v.a. (H_i, B_i, Φ_i) para la i-ésima ronda. Utilizando el Teorema de la probabilidad compuesta, y observando la dependencia de las variables durante la ejecución del protocolo, calculamos:

$$P(H_i = h, B_i = b, \Phi_i = \pi) =$$
$$P(\Phi_i = \pi) \cdot P(B_i = b \mid \Phi_i = \pi) \cdot P(H_i = h \mid \Phi_i = \pi, B_i = b)$$

El isomorfismo π se elige aleatoriamente entre todas las posibles permutaciones de V_0, como $\mid Sym(V_0) \mid = n!$, $P(\Phi_i = \pi) = \frac{1}{n!}$.

La variable aleatoria B_i se calcula con la función F, $B_i = F(h, V_i, H_i)$, por lo que asignamos la probabilidad $P(B_i = b \mid \Phi_i = \pi) = p_b$ dependiente de la F que use V.

Por último $P(H_i = h \mid \Phi_i = \pi, B_i = b) = 1$ por construcción del grafo h por el isomorfismo π, independiente del valor de B_i.

Nos queda en total que $P(H_i = h, B_i = b, \Phi_i = \pi) = \frac{p_b}{n!}$.

Ahora consideramos la tupla de v.a. (H_i', B_i', Φ_i') del Simulador.

La variable Φ_i' se elige aleatoriamente entre $Sym(V_0)$ o $Sym(V_1)$, ambos de mismo orden pues $\mid V_0 \mid = \mid V_1 \mid = n$, por lo que $\mid Sym(V_0) \mid = \mid Sym(V_1) \mid = n!$, luego obtenemos $P(\Phi_i' = \pi) = \frac{1}{n!}$.

Como en la Vista, el Simulador utiliza F para calcular el valor b de B_i', así que $P(B_i' = b \mid \Phi_i' = \pi) = p_b$.

Finalmente, h viene determinado por π, de modo que $P(H_i' = h \mid \Phi_i' = \pi, B_i' = b) = 1$.

La probabilidad del Simulador nos queda $P(H_i' = h, B_i' = b, \Phi_i' = \pi) = \frac{p_b}{n!}$, igual que la del ensamble de la Vista.

Concluimos que se cumple la propiedad de *conocimiento cero* y el problema GI tiene una prueba interactiva de conocimiento cero perfecta.

□

5.2.3 *Prueba de conocimiento cero para el problema del logaritmo discreto*

La última prueba de conocimiento cero perfecta que veremos será la del problema del logaritmo discreto (Sección 4.4) en su primera versión, el grupo es $G = \langle g \rangle$ de orden primo q, donde P debe probar que conoce la potencia s que genera un cierto valor $y = g^s$.

Teorema 5.13. *El problema DL tiene una prueba de conocimiento cero.*

Demostración.

Primero veremos un protocolo interactivo entre P y V para probar el conocimiento de s, y entonces comprobaremos las tres propiedades necesarias, completitud, robustez y conocimiento cero.

Algoritmo 5.14 (Prueba interactiva para DL)**.**

Datos comunes: Una instancia (G, q, g, y) del Problema DL. $n = \mathrm{ord}(g) = q$ es el tamaño del problema.

Protocolo: Sea $t(n)$ un polinomio en n. P y V repiten $t(n)$ veces los siguientes pasos.

1. P elige aleatoriamente $u \in_R \mathbb{Z}_q^*$.
2. $P \to V: \quad a = g^u$.
3. $V \to P: \quad b \in_R \{0, 1\}$.
4. $P \to V: \quad w = (u + sb) \bmod q$.
5. V comprueba si:

$$g^w \stackrel{?}{=} \begin{cases} a & \text{si } b = 0 \\ a \cdot y & \text{si } b = 1. \end{cases}$$

   Si la comparación falla, V termina en rechazo. En caso contrario, vuelve al paso 1.

Tras $t(n)$ rondas, V termina y acepta la instancia como $\mathtt{Verdadera}$, P conoce el logaritmo discreto de $y \in G = \langle g \rangle$.

Nótese que por el propio enunciado del problema, P no ha necesitado de su potencia ilimitada de cálculo.

El protocolo es *completo*, pues si P conoce s, siempre puede calcular un w que pase la comprobación de V en el paso 5.

Es también *robusto*, pues suponiendo un grupo G donde un P tramposo, una máquina limitada a cálculos en tiempo polinomial, no puede calcular fácilmente el secreto s, este P tramposo deberá intentar adivinar el reto b.

Si P^* supone que el reto $b = 0$, seguirá el protocolo, mandando $w = u$, pero fallaría si V manda $b = 1$, al no poder calcular el s.

Si P^* quiere superar un reto $b = 1$, puede elegir u como en el paso 1, enviar $a = g^u \cdot y - 1$, y contestar al reto con $w = u$. Si se equivoca y V envía $b = 0$, necesitaría s para poder enviar el w correcto y fallaría la prueba.

La probabilidad de acertar el reto b es de $1/2$, de modo que la probabilidad de pasar la prueba interactiva con una instancia $\mathtt{Falsa}$ es de $2^{-t(n)}$. Se cumple la propiedad de *robustez*.

Nos queda comprobar la propiedad de *conocimiento cero*.

Como en casos anteriores, suponemos un algoritmo probabilístico polinomial F que utiliza V^* para enviar los retos. Si V fuera honesto, F es un generador de números aleatorios con una distribución uniforme.

Simulador para el problema DL $S_{V^*}(G, q, g, y, h)$.

Datos: (G, q, g, y), una instancia $Verdadera$ del problema DL; h, transcripciones de ejecuciones previas del protocolo; v_i, transcripción de la interacción actual (i rondas).
Ejecución: Repetir para $i+1 \leqslant t(n)$:

1. Elegir $b_{i+1} \in_R \{0, 1\}$
2. Elegir $w_{i+1} \in_R \mathbb{Z}_q^*$.
3. **Si** $b_{i+1} = 0$, **entonces** calcular $a_{i+1} = g^w$
   Si no, $a_{i+1} = g^w y^{-1} = g^{w-s}$
4. **Si** $b_{i+1} = F(G, q, g, y, h, v_i, a_{i+1})$, **entonces** añadir la tupla $(a_{i+1}, b_{i+1}, w_{i+1})$ a la transcripción.
   Si no, volver al paso 1.
5. $i = i+1$

Por la elección aleatoria de b_{i+1} y w_{i+1} el Simulador es un algoritmo probabilístico, y el tiempo de ejecución estimado es de dos iteraciones por ronda simulada, pues la probabilidad de que el b_{i+1} elegido coincida con el indicado por F es de $1/2$.

Por inducción sobre i, el número de rondas realizadas, veamos que los ensambles $Vista_{P,V^*}$ y S_{V^*} son iguales:

$$Vista_{P,V^*} = (G, q, g, y, h, A_1, B_1, W_1, \ldots, A_i, B_i, W_i)$$
$$S_{V^*} = (G, q, g, y, h, A_1', B_1', W_1', \ldots, A_i', B_i', W_i'),$$

donde las v.a. A_i, A_i' representan el testigo a_i de la i-ésima ronda, las v.a. B_i, B_i' el bit del reto, y W_i, W_i' la respuesta de P.

Para $i = 0$, los ensambles $Vista_{P,V^*} = (G, q, g, y, h)$ y $S_{V^*} = (G, q, g, y, h)$ son constantes e iguales.
Suponemos cierto para $i-1$.
La tupla de las v.a. para la ronda i en la Vista es: (A_i, B_i, W_i).
Su probabilidad se calcula como:

$$P(A_i = a, B_i = b, W_i = w) =$$
$$P(A_i = a) \cdot P(B_i = b \mid A_i = a) \cdot P(W_i = w \mid A_i = a, B_i = b)$$

Por construcción, a depende de la elección de u, elegido de entre $q-1$ posibles valores, $P(A_i = a) = 1/(q-1)$.
V^* utiliza $F(G, q, g, y, a)$ para la elección de b, así que podemos escribir $P(B_i = b \mid A_i = a) = p_b$, dependiente de la F empleada.
Finalmente, en el protocolo w depende para su cálculo de a y b, así que la probabilidad dependiente de ambos valores dados, es 1.
Concluimos que $P(A_i = a, B_i = b, W_i = w) = \dfrac{p_b}{q-1}$.

Observando el orden en que se calculan los valores en el Simulador, la probabilidad de la i-ésima tupla del Simulador es:

$$P(A_i' = a, B_i' = b, W_i' = w) =$$
$$P(W_i' = w) \cdot P(B_i' = b \mid W_i' = w) \cdot P(A_i' = a \mid W_i' = w, B_i' = b)$$

El valor de w_{i+1} se elige en $\mathbb{Z}_q^*$ independientemente de los demás. Así que $P(W_i' = w) = 1/(q-1)$.

Como antes, la elección del bit depende de F. $P(B_i' = b \mid W_i' = w) = p_b$.

Y siguiendo los pasos del Simulador, el valor de a queda unívocamente determinado dados w y b, así que $P(A_i' = a \mid W_i' = w, B_i' = b) = 1$.

Nos queda $P(A_i' = a, B_i' = b, W_i' = w) = \dfrac{p_b}{q-1}$, igual que la Vista, de modo que los ensambles son idénticos y queda demostrado el teorema.

□

OTROS TIPOS DE PRUEBAS DE CONOCIMIENTO CERO La propiedad de *conocimiento cero perfecta* exige que exista un Simulador probabilístico de tiempo polinomial, cuyo ensamble sea idéntico al del protocolo. Una variación de la propiedad de conocimiento cero admite que los ensambles sean asintóticamente iguales, y de este modo a las pruebas se les llama *pruebas de conocimiento cero estadísticas*.

Al relajar la definición de conocimiento cero, ampliamos la cantidad de pruebas de conocimiento cero conocidas, y por eso se definen diferentes tipos de *conocimiento cero*, con menos restricciones, que en la práctica pueden funcionar como las pruebas de conocimiento cero perfectas, e incluso en ocasiones ser más óptimos.

5.3 PRUEBAS DE CONOCIMIENTO CERO DE VERIFICADOR HONESTO

En esta sección veremos un tipo de pruebas de conocimiento cero utilizadas en la práctica, que derivan de las ZKP perfectas al añadir una condición al Verificador: que siempre cumpla el protocolo indicado.

En las pruebas de conocimiento cero perfectas que hemos visto, los simuladores estudiados consideraban la existencia de un algoritmo F que un Verificador tramposo, V^*, utilizaría para elegir los retos, en un intento de obtener más información de P. En este tipo de pruebas, V no dispondrá de F para elegir los retos.

Definición 5.15 (Propiedad de conocimiento cero con Verificador Honesto)**.**

Un sistema de prueba interactiva (completo y robusto), para un problema de decisión Q, es *perfecta de conocimiento cero con Verificador Honesto* si el ensamble $\text{Vista}_{P,V}(q, h)$ es idéntico al ensamble generado por un Simulador $S_V(q, h)$, donde el Verificador V sigue los pasos del protocolo, para cualquier instancia Verdadera $q \in Q$ y cualquier historial h.

Veremos en este apartado una variación de la prueba de conocimiento cero perfecta basada en el problema del Logaritmo Discreto (4.4), llamado *Protocolo de Identificación de Schnorr*. Para esta prueba seguiremos principalmente el texto de Stinson [19, Section 9.4].

Primero hemos de definir unos parámetros comunes conocidos por P y V. Elegimos dos valores primos, p y q tal que $p \equiv 1 \bmod q$, es decir, $q \mid (p-1)$, y el problema del logaritmo discreto es difícil en el subgrupo de $\mathbb{Z}_p^*$ generado por α con orden q. Además debemos elegir un valor t que definirá la robustez del protocolo. En resumen tenemos:

- p primo.
- q primo divisor de $(p-1)$.
- $\alpha \in \mathbb{Z}_p^*$ de orden q.
- t un *parámetro de seguridad*, con la condición de que $2^t < q$. La robustez del protocolo, es decir, la probabilidad de que un P* tramposo engañe a V, será de 2^{-t}.

El *secreto* de P será el valor a, tal que $0 \leqslant a \leqslant q-1$. La prueba consistirá en demostrar que P conoce dicho valor. P calcula entonces el valor público $v = \alpha^{-a} \bmod p$. Éste valor se puede calcular como la inversa $(\alpha^a)^{-1} \bmod p$, o como $\alpha^{q-a} \bmod p$.

Mostramos el protocolo de Schnorr:

Algoritmo 5.16 (Schnorr)**.**
Datos comunes: p, q, α, t y v.
Datos de P: a.
Protocolo: Realizar una vez:

1. P elige aleatoriamente $k \in_R \mathbb{Z}_q$.
2. $P \to V: \quad \gamma = \alpha^k \bmod p$ (el *testigo*).
3. $V \to P: \quad r$ aleatorio, tal que $1 \leqslant r \leqslant 2^t < q$ (el *reto*).
4. $P \to V: \quad y = k + ar \bmod q$ (la *respuesta*).
5. V comprueba si $\gamma \stackrel{?}{\equiv} \alpha^y v^r \bmod p$

   Si la comparación falla, V termina en rechazo. En caso contrario, acepta la prueba.

Este algoritmo en particular se diseñó con el objetivo de minimizar la cantidad de mensajes intercambiados manteniendo una robustez que lo hiciera seguro. Como veremos, una sola interacción nos dará una probabilidad de 2^{-t} de que un atacante consiga engañar al Verificador.

Teorema 5.17. *El protocolo de identificación de Schnorr 5.16 es una prueba interactiva.*

Demostración. Como sólo existe una ronda de 3 mensajes, el algoritmo es probabilístico de tiempo polinomial.

Veamos ahora la *completitud*. Veamos con las siguientes congruencias que que si P conoce a, siempre podrá responder correctamente al reto:

$$\alpha^y v^r \equiv \alpha^{k+ar} v^r \equiv \alpha^{k+ar} \alpha^{-ar} \equiv \alpha^k \equiv \gamma \bmod p$$

Por tanto, un P y V honestos finalizarán el protocolo en aceptación siempre, la prueba es *completa*.

Finalizamos con el análisis de la *robustez*. El *parámetro de seguridad* t define la dificultad de adivinar el reto de V. Si P^* adivinara el valor r aleatorio de V, podría superar la prueba eligiendo un valor y aleatorio y calculando $\gamma = \alpha^y v^r \bmod p$, con lo que enviando primero el y y respondiendo al reto con γ pasaría la prueba. La probabilidad de adivinar correctamente un reto $1 \leqslant r \leqslant 2^t$ elegido aleatoriamente es de $\frac{1}{2^t}$.

Supongamos que P^* puede adivinar el reto de V con una probabilidad mayor a 2^{-t}, entonces P^* conocería para un valor γ de su elección, al menos dos respuestas posibles para dos retos distintos de V, es decir, si antes el P^* tramposo podía precalcular un testigo y una respuesta para un reto de V, ahora puede calcular, para el mismo testigo (pues solo puede enviar uno antes de recibir el reto), P^* conoce la respuesta de al menos dos retos.

Sea γ el testigo para el que P^* conoce los valores r_1, y_1 y r_2, y_2, un par de posibles retos y respuestas válidas, que cumplen $\gamma \equiv \alpha^{y_1} v^{r_1} \equiv \alpha^{y_2} v^{r_2} \bmod p$. Se sigue que $\alpha^{y_1 - y_2} \equiv v^{r_2 - r_1} \bmod p$.

Como la *clave pública* se calculaba $v \equiv \alpha^{-a} \bmod p$, con a el secreto de P, podemos sustituir: $\alpha^{y_1 - y_2} \equiv \alpha^{-a(r_2 - r_1)} \bmod p$. Y como α tiene orden q, podemos trabajar con los exponentes como $y_1 - y_2 \equiv a(r_1 - r_2) \bmod q$.

Por la condición sobre los retos $1 \leqslant r \leqslant 2^t < q$, sabemos que $0 <| r_2 - r_1 |< 2^t < q$, y como q es primo, $\mathrm{mcd}(r_2 - r_1, q) = 1$, de modo que existe el inverso $(r_1 - r_2)^{-1} \bmod q$, y el P^* puede calcular el secreto a como $a = (y_1 - y_2)(r_1 - r_2)^{-1} \bmod q$.

A partir de este análisis, vemos que cualquiera que pueda superar la prueba frente a V con una probabilidad mayor a 2^{-t} puede calcular el secreto de P, y sería una instancia $\mathit{Verdadera}$ del problema.

□

Después de ver la *completitud* y *robustez* del protocolo, tenemos una prueba interactiva, pero nos falta comprobar la propiedad de *conocimiento cero con verificador honesto*.

Teorema 5.18. *La prueba interactiva de Schnorr 5.16 es de conocimiento cero con verificador honesto.*

Demostración. Una transcripción o vista del protocolo es simplemente el ensamble probabilístico con una tupla $\mathit{Vista}_{P,V} = (\Gamma, R, Y)$.

Calculamos la probabilidad $P(\Gamma = \gamma, R = r, Y = y) = P(\Gamma = \gamma)P(R = r \mid \Gamma = \gamma)P(Y = y \mid \Gamma = \gamma, R = r)$.

P calcula γ a partir de un valor k aleatorio entre 0 y $q - 1$, y elevando α, de orden q, a dicho valor, de modo que la probabilidad de que Γ tome el valor γ es igual a la de elegir el k que lo genera: $P(\Gamma = \gamma) = \frac{1}{q}$.

Bajo el supuesto de Verificador Honesto, $P(R = r \mid \Gamma = \gamma) = P(R = r) = \frac{1}{2^t}$, pues V seguirá el protocolo, eligiendo el reto uniformemente entre 1 y 2^t, sin usar la función F de apartados previos.

Finalmente, $y = k + ar$ depende del k que genera $\Gamma = \gamma$, del secreto a de P y del reto $R = r$ de V. La probabilidad condicionada vale $P(Y = y \mid \Gamma = \gamma, R = r) = 1$.

Tenemos que la probabilidad de obtener la Vista (γ, r, y) es de $\frac{1}{q \cdot 2^t}$.

Ahora definimos el Simulador de la prueba, con la restricción de Verificador Honesto:

Algoritmo 5.19.
Datos: p, q, α, t y v.
Protocolo: Realizar una vez:

1. Elegir r aleatorio tal que $1 \leqslant r \leqslant 2^t$.
2. Elegir y aleatorio tal que $0 \leqslant y \leqslant q-1$.
3. Calcular $\gamma = \alpha^y v^r \bmod p$.

El ensamble del Simulador se puede escribir como el conjunto:

$$S_V = (\Gamma, R, Y) = \{(\gamma', r', y') : 1 \leqslant r' \leqslant 2^t,\ 0 \leqslant y' \leqslant q-1,\ \gamma' \equiv \alpha^y v^r \bmod p\}$$

Como γ' depende totalmente de r' e y', que se eligen independientemente entre sí, tenemos que $\mid S_V \mid = q \cdot 2^t$, es decir, $P(\Gamma' = \gamma', R' = r', Y' = y) = \frac{1}{q \cdot 2^t}$, la misma que la probabilidad de la Vista.

Tenemos por tanto, que el protocolo de Schnorr es una Prueba de Conocimiento Cero con Verificador Honesto.

□

A día de hoy, para probar que Schnorr es de conocimiento cero perfecta, donde un Verificador utilizaría la función F que elige los retos no uniformemente entre 1 y 2^t, no se conoce ningún simulador que sea probabilístico en tiempo polinomial, condición indispensable para probar que es perfecta.

Aún así, Schnorr es utilizado ampliamente en la práctica, pues tampoco se conoce ningún ataque a partir de la elección de retos no aleatorios. Se puede demostrar que algunas variaciones del protocolo son pruebas de conocimiento cero perfectas, para ello hay que añadir ciertas condiciones, como reducir el espacio de retos, a cambio de incrementar las rondas para mantener el nivel de robustez. Con un bit por reto y t rondas, en la sección anterior demostramos que el protocolo era una prueba de conocimiento cero perfecta, pero en la práctica es muy poco eficiente por la cantidad de mensajes intercambiados.

5.4 ESQUEMAS DE COMPROMISO

En las pruebas de conocimiento cero vistas hasta ahora, el primer mensaje del Probador consistía en un *compromiso* con el Verificador, al que llamamos *testigo*, de modo que en la respuesta al reto no pudiera engañarle, con cierta probabilidad.

Antes de introducir el último tipo de Pruebas de Conocimiento Cero de este capítulo, las Computacionales, debemos estudiar la existencia y propiedades de los llamados esquemas de compromiso, una herramienta que nos permite dar para todo problema **NPC** una Prueba de Conociemiento Cero Computacional, como muestra Manuel Blum en "How to Prove a Theorem So No One Else Can Claim It" [3]. La bibliografía básica para esta sección se encuentra principalmente en [11, 16].

Estos esquemas permiten esconder la estructura de una instancia $Verdadera$, donde P se compromete con su *testigo* antes de conocer el *reto* de V, de modo que V no puede obtener información de estos testigos. En la última *respuesta* de P, se revelará una pequeña parte de la estructura de la instancia $Verdadera$, de la cual V no podrá aprender tampoco nada.

Empecemos definiendo un esquema de compromiso para un bit:

Definición 5.20. Un esquema de compromiso de un *bit* es un par de funciones (f, v) de tiempo polinomial. La función

$f : \{0,1\} \times \Upsilon \to \chi$

transforma el bit $b \in \{0,1\}$ con una clave aleatoria $y \in \Upsilon$. El valor $x = f(b, y)$ lo llamaremos *blob* o *testigo* de b. La función de verificación

$v : \chi \times \Upsilon \to \{0, 1, \bullet\}$

abre el blob revelando el bit b, o indicando que no es un par (blob,clave) válido.

El par (f, v) debe cumplir la siguientes condiciones:

1. *Vinculación*: Para todo blob $x = f(b, y)$, P no es capaz de encontrar un valor $y' \neq y$ tal que el blob se puede abrir con otro valor, es decir, $v(x, y) \neq v(x, y')$.
2. *Secreto*: Los ensambles $\{f(0, \Upsilon)\}$ y $\{f(1, \Upsilon)\}$ son indistinguibles.

Podemos clasificar los esquemas de compromiso de bit en dos tipos:

Definición 5.21 (Vinculación incondicional)**.** Una máquina P con una capacidad computacional ilimitada (como en las pruebas interactivas) no puede cambiar el bit comprometido en el *testigo* enviado.

Definición 5.22 (Secreto incondicional)**.** Los ensambles $\{f(0, \Upsilon)\}$ y $\{f(1, \Upsilon)\}$ son idénticos. Es decir, dada una máquina V con capacidad computacional ilimitada, no es capaz de distinguir los ensambles con mayor probabilidad que eligiendo al azar el posible valor de b.

Proposición 5.23. *En un entorno donde tenemos dos interlocutores, P y V, que ven todo lo que el otro les envía, no existe ningún esquema de compromiso con vinculación y secreto incondicionales a la vez.*

Demostración. Supongamos que tenemos vinculación incondicional, P no puede encontrar un valor y' que abra x con un bit diferente. Entonces V podría distinguir a qué b corresponde el testigo x recibido, siendo un posible algoritmo probar todos los valores $y' \in \Upsilon$ hasta encontrar el que hace $v(x, y') \neq \bullet$. Aunque una máquina convencional podría tardar demasiado tiempo, estamos suponiendo que V puede ser computacionalmente ilimitado para el caso de secreto incondicional, y por tanto, no podemos tener ambas propiedades incondicionales a la vez. □

Sin embargo, se puede conseguir una condición menos restrictiva, pero equivalentes en la práctica:

Definición 5.24 (Vinculación computacional)**.** No existe ningún algoritmo probabilístico en tiempo polinomial que permita encontrar un y' tal que P puede cambiar el bit comprometido.

Definición 5.25 (Secreto computacional)**.** No existe ningún algoritmo probabilístico en tiempo polinomial que permita distinguir los ensambles probabilísticos $\{f(0, \Upsilon)\}$ y $\{f(1, \Upsilon)\}$. Se dice que los ensambles son *indistinguibles polinomialmente*.

Para conseguir vinculación y secreto incondicionales a la vez, han de utilizarse otras técnicas de comunicación, como canales con ruido o múltiples participantes [2, 7, 9-11].

5.4.1 *Esquemas de compromiso con secreto incondicional*

En este apartado vamos a mostrar ejemplos de esquemas de compromiso de bit que consiguen secreto incondicional y vinculación computacional, basándose en los mismos problemas vistos para las pruebas de conocimiento perfectas: residuos cuadráticos, logaritmo discreto e isomorfismo de grafos.

El esquema basado en residuos cuadráticos lo inicia V, eligiendo el módulo $n = pq$ producto de dos primos, de modo que el problema QR módulo n es impracticable para P. V además elige un $t \in_R \mathbb{Z}_n^*$ para calcular el residuo $s = t^2 \bmod n$. El par (s, n) se hace público para que P lo utilice. Utilizaremos como conjunto de *claves* $\Upsilon = \mathbb{Z}_n^*$ y como conjunto de los posibles blobs o testigos $\chi = \mathbb{Z}_n^{Q+}$, los residuos cuadráticos.

El algoritmo de compromiso es como sigue:

Algoritmo 5.26 (Compromiso de bit basado en residuos cuadráticos)**.**

Ocultación: P elige al azar $y \in \mathbb{Z}_n^*$ y oculta el bit b como:

$$x = f(b, y) = s^b y^2 \bmod n$$

Apertura: P envía a V el valor y. V calcula el bit b:

$$b = v(x, y) = \begin{cases} 0 & \text{si } x \equiv y^2 \bmod n \\ 1 & \text{si } x \equiv s \cdot y^2 \bmod n \\ \bullet & \text{en otro caso.} \end{cases}$$

Proposición 5.27. *El algoritmo 5.26 es un esquema de compromiso de bit con vinculación computacional y secreto incondicional.*

Demostración. La ocultación consiste en enviar como testigo un residuo cuadrático módulo n, y a la hora de desvelar el valor del bit, P debe desvelar una raíz cuadrada de x. Si P es una máquina limitada polinomialmente, al no conocer la factorización de n, no podrá calcular la raíz cuadrada de x. Si pudiera calcular dicha raíz, el ataque consistiría en seguir el protocolo para el bit $b = 1$, de modo que conocemos el y para abrirlo como 1, y cuando P quiera cambiar el bit oculto a 0, calculará la raíz cuadrada módulo n de $s \cdot y^2$, de modo que V obtendrá en la *apertura* $b = 0$. Suponiendo que el problema de la factorización es intratable, tenemos **vinculación computacional**.

Por otro lado, sea $\beta = |\ \Omega_x\ |$ con Ω_x el conjunto de raíces modulares de x, que un V *todopoderoso* es capaz de calcular. Sea $r \in \Omega_x$ la raíz modular de x usada por P en el protocolo, e Y la variable aleatoria que representa la clave y. Si $b = 0$ tenemos que $P(Y = y) = P(r = y) = 1/\beta$, pues alguna de las raíces será igual a y. Si $b = 1$, la raíz utilizada para generar x será $r = ty \bmod n$, y como t es un valor prefijado, y está unívocamente determinada por r, entonces $P(Y = y) = P(r = ty) = 1/\beta$. Por tanto, los ensambles probabilísticos $\{f(0, \Upsilon)\}$ y $\{f(1, \Upsilon)\}$ son idénticos, y tenemos **secreto incondicional**.

□

A continuación, presentamos otro esquema de compromiso basado en el logaritmo discreto, conocido como *compromiso de Pedersen*. Para llevarlo a cabo, P y V deben primero decidir un p primo suficientemente grande y un generador $g \in \mathbb{Z}_p^*$ donde el problema del logaritmo discreto es *difícil*. V además elige un valor aleatorio $s \in \mathbb{Z}_{p-1}$, de modo que no se conoce $\log_g s$, y lo envía a P. Los conjuntos del esquema de compromiso son $\Upsilon = \mathbb{Z}_{p-1}$ para las *claves* y $\chi = \mathbb{Z}_p^*$ para los blobs.

Algoritmo 5.28 (Compromiso de bit basado en logaritmo discreto)**.**
Ocultación: P elige al azar $y \in \Upsilon$ y oculta el bit b como:

$$x = f(b, y) = s^b g^y \bmod p$$

Apertura: P envía a V el valor y. V calcula el bit b:

$$b = v(x, y) = \begin{cases} 0 & \text{si } x \equiv g^y \bmod p \\ 1 & \text{si } x \equiv sg^y \bmod p \\ \bullet & \text{en otro caso.} \end{cases}$$

Proposición 5.29. *El algoritmo 5.28 es un esquema de compromiso de bit con vinculación computacional y secreto incondicional.*

Demostración. El ataque a este esquema de compromiso consistiría en calcular el logaritmo discreto de s módulo p, llamémoslo t, tal que $g^t = s \bmod p$. Si P oculta $b = 0$, enviará $x = g^y \bmod p$, y para poder abrir al valor $b = 1$ puede enviar a V $(y - t)$, de modo que V calculará $sg^{y-t} \equiv g^t g^{y-t} \equiv g^y \equiv x \bmod p$, obteniendo $b = 1$. En el caso contrario, donde P oculta $b = 1$, enviará $x = sg^y \bmod p$ a V como testigo, y para abrir el bit como $b = 0$ basta con enviar $y + t$, tal que V calculará $g^{y+t} \equiv g^t g^y \equiv sg^y \bmod p$, creyendo entonces que $b = 1$. Si P es computacionalmente limitado, no podrá resolver el problema del logaritmo discreto y obtenemos una **vinculación computacional**.

Suponiendo un V computacionalmente ilimitado, éste podría calcular el logaritmo discreto de x módulo p, llamémoslo u, tal que $x \equiv g^u \bmod p$, además del valor t que genera s, $s \equiv g^t \bmod p$. Una vez conocidos u y t, la probabilidad de que y valga u ó $(u - t)$, según sea $b = 0$ $(x \equiv g^u \equiv g^y \bmod p)$ ó $b = 1$ $(x \equiv g^t g^{u-t} \equiv sg^y \bmod p)$ respectivamente, es la misma, $1/2$. Tenemos entonces que los ensambles $\{f(0, \Upsilon)\}$ y $\{f(1, \Upsilon)\}$ son idénticos y el esquema es de **secreto incondicional**.

□

Finalmente utilizamos el problema del isomorfismo de grafos para mostrar otro esquema de secreto incondicional. En este caso P y V escogen un grafo $G = (V, E)$ $(n = |V|)$. V escoge al azar una permutación $\pi \in_R Sym(V)$ del grafo y calcula $H = \pi(G)$. El par de grafos (G, H) lo conocerán tanto P como V, pero la permutación π la mantendrá en secreto

V. Los conjuntos del esquema de compromiso son $\Upsilon = Sym(V)$ y $\chi = \{H \mid H = \pi(G), \pi \in_R Sym(V)\}$.

Algoritmo 5.30 (Compromiso de bit basado en isomorfismo de grafos)**.**
Ocultación: P elige al azar $\gamma \in \Upsilon$ y oculta el bit b como:

$$X = f(b, \gamma) = \begin{cases} \gamma(G) & \text{si } b = 0 \\ \gamma(H) & \text{si } b = 1 \end{cases}$$

Apertura: P envía a V el valor γ. V calcula el bit b:

$$b = v(X, \gamma) = \begin{cases} 0 & \text{si } \gamma(G) = X \\ 1 & \text{si } \gamma(H) = X \\ \bullet & \text{en otro caso.} \end{cases}$$

Proposición 5.31. *El algoritmo 5.30 es un esquema de compromiso de bit con vinculación computacional y secreto incondicional.*

Demostración. P podría atacar el esquema si conociera el isomorfismo de G a H, componiéndolo con γ para cambiar el bit que V abriría. Como suponemos que P es una máquina limitada computacionalmente, no puede resolver el problema del isomorfismo de grafos y por tanto el esquema de compromiso es de **vinculación computacional**.

Además, como los posibles γ para abrir un bit son permutaciones aleatorias de G, compuestas o no con π, la probabilidad de que dado un X el bit b sea 0 ó 1 es 1/2 en ambos casos, de modo que los ensambles $\{f(0, \Upsilon)\}$ y $\{f(1, \Upsilon)\}$ son idénticos, y tenemos **secreto incondicional**. □

5.4.2 *Esquemas de compromiso con vinculación incondicional*

Podemos conseguir esquemas de compromiso con vinculación incondicional a costa de obtener secreto computacional, como demostramos antes, no podemos obtener ambas propiedades a la vez de modo incondicional. Veremos en este apartado un ejemplo basado en residuos cuadráticos.

Partiendo del problema del residuo cuadrático, para $N = pq$, los conjuntos $\mathbb{Z}_N^{Q+}$ y $\mathbb{Z}_N^{Q-}$ son polinomialmente indistinguibles. En este caso, P inicia el esquema eligiendo 2 primos aleatorios suficientemente grandes, p y q, y un **no**-residuo cuadrático $s \in \mathbb{Z}_N^{Q-}$, es decir, con símbolo de Jacobi 1. El par (s, N) se hace público para V. El conjunto de *claves* del esquema es $\Upsilon = \mathbb{Z}_N^*$, y el de posibles valores de los blobs es $\chi = \mathbb{Z}_N^Q$, los valores con símbolo de Jacobi 1, sea o no residuo cuadrático.

Algoritmo 5.32 (Compromiso de bit basado en residuos cuadráticos)**.**
Ocultación: P elige al azar $y \in \mathbb{Z}_N^*$ y oculta el bit b como:

$$x = f(b, y) = s^b y^2 \bmod N$$

Apertura: P envía a V el valor y. V calcula el bit b:

$$b = v(x, y) = \begin{cases} 0 & \text{si } x \equiv y^2 \bmod N \\ 1 & \text{si } x \equiv s \cdot y^2 \bmod N \\ \bullet & \text{en otro caso.} \end{cases}$$

Proposición 5.33. *El algoritmo 5.32 es un esquema de compromiso de bit con secreto computacional y vinculación incondicional.*

Demostración. El valor del testigo x será un residuo cuadrático si $b = 0$, o un no-residuo cuadrático si $b = 1$, y como $\mathbb{Z}_N^{Q+} \cap \mathbb{Z}_N^{Q-} = \emptyset$, P es incapaz de abrir el testigo con otro valor de b, aunque sea computacionalmente ilimitado. Tenemos por tanto **vinculación incondicional**.

Sin embargo, los ensambles $\{f(0, \Upsilon)\}$ y $\{f(1, \Upsilon)\}$ son sólo polinomialmente indistinguibles, bajo la suposición de que el problema QR es *difícil*, de modo que obtenemos sólo **secreto computacional**. □

Este esquema es casi idéntico al 5.26, pero la diferencia radica en que en el esquema con secreto incondicional, sólo usamos residuos cuadráticos, de modo que no se podía distinguir una apertura de otra hasta que la enviara P, y en este caso utilizamos un no-residuo cuadrático con símbolo de Jacobi 1, y en vez de secreto incondicional, conseguimos vinculación incondicional.

5.4.3 *Esquemas de compromiso para cadenas de bits*

Existen pruebas interactivas donde P necesita comprometerse con un valor en un conjunto mayor que $\{0, 1\}$, de modo que para ocultarlo, con las técnicas previas, debería representar en base binaria dicho valor y ejecutar el esquema de compromiso bit a bit, lo cual es claramente ineficiente. Vamos a ampliar la definición 5.20:

Definición 5.34. Un esquema de compromiso de un cadenas de n bits es un par de funciones (f, v) de tiempo polinomial. La función
$f : \{0, 1\}^n \times \Upsilon \to \chi$
transforma la cadena $s = (b_1, \ldots, b_n) \in \{0, 1\}^n$ con una clave aleatoria $y \in \Upsilon$. El valor $x = f(b, y)$ lo llamaremos *blob* o *testigo* de s. La función de verificación
$v : \chi \times \Upsilon \to \{0, 1, \ldots, 2^{n-1} \bullet\}$
abre el blob revelando la cadena s, o indicando que no es un par (blob,clave) válido.

Veamos dos ejemplos basados en el problema del logaritmo discreto. El primero se basa en el compromiso de Pedersen, siendo de secreto incondicional, mientras que en el segundo ejemplo, basado en el algoritmo de ElGamal, conseguimos vinculación incondicional.

Comencemos por el compromiso de Pedersen. Como antes, P y V deben elegir un primo p suficientemente grande para que el problema del logaritmo discreto sea impracticable. Además, eligen un generador $g \in \mathbb{Z}_p^*$ y un valor aleatorio h tal que no se conoce $\log_g h$. El conjunto de *claves* será $\Upsilon = \mathbb{Z}_{p-1}$ y el de blobs $\chi = \mathbb{Z}_p^*$. El compromiso de una cadena s se obtiene como sigue:

Algoritmo 5.35.
Ocultación: P elige al azar $y \in \Upsilon$ y oculta s como:

$$x = f(s, y) = g^s h^y \bmod p$$

Apertura: P envía a V el par (s', y'). V comprueba que

$$s = \nu(x, y) = \begin{cases} s' & \text{si } x \equiv g^{s'} h^{y'} \bmod p \\ \bullet & \text{en otro caso.} \end{cases}$$

Proposición 5.36. *El algoritmo 5.35 es un esquema de compromiso de cadena de bits con vinculación computacional y secreto incondicional.*

Demostración. Para poder romper la propiedad de vinculación, haría falta encontrar los pares (s_0, y_0) y (s_1, y_1), con $s_0 \neq s_1$, tal que

$$g^{s_0} h^{y_0} \equiv g^{s_1} h^{y_1} \bmod p$$

operando obtenemos $g^{s_0 - s_1} \equiv h^{y_1 - y_0} \bmod p$, y podemos despejar

$$h \equiv g^{(s_0 - s_1)(y_1 - y_0)^{-1} \bmod p-1} \bmod p$$

y habríamos calculado el logaritmo discreto de h en base g, pero para un P computacionalmente limitado, y suponiendo que el problema del logaritmo discreto es impracticable, esto no es posible, por lo que obtenemos **vinculación computacional**.

Por otra parte, para cualquier par (s, y) y t tal que $h \equiv g^t \bmod p$, para cualquier posible valor s' tenemos que $y' = \frac{s - s'}{t} + y$ cumple $g^s h^y \equiv g^{s'} h^{y'} \bmod p$. Es decir, s' se puede abrir con la *clave* $y' = \frac{s - s'}{t} + y$, donde y se elige uniformemente en $\mathbb{Z}_p^*$. Tenemos entonces que el esquema es de **secreto incondicional**.

□

Ahora veamos el segundo ejemplo de compromiso de cadenas de bits. La inicialización es la misma que en el caso anterior, P y V conocen un primo p, un generador $g \in \mathbb{Z}_p^*$

y un valor aleatorio $h \in \mathbb{Z}_p^*$ tal que no se conoce $\log_b h$. Los conjuntos del esquema de compromiso son $\Upsilon = \mathbb{Z}_{p-1}$ y $\chi = \mathbb{Z}_p^*$.

Algoritmo 5.37.

Ocultación: P elige al azar $y \in \Upsilon$ y oculta s como el par de valores:

$$(x_1, x_2) = f(s, y) = (g^y \bmod p,\ g^s h^y \bmod p)$$

Apertura: P envía a V el par (s', y'). V comprueba que

$$s = v(x, y) = \begin{cases} s' & \text{si } x_1 \equiv g^{y'} \bmod p \quad y \quad x_2 \equiv g^{s'} h^{y'} \bmod p \\ \bullet & \text{en otro caso.} \end{cases}$$

Proposición 5.38. *El algoritmo 5.35 es un esquema de compromiso de cadena de bits con vinculación computacional y secreto incondicional.*

Demostración. En este caso, el valor de $x_1 = g^y \bmod p$ determina unívocamente el valor de y elegido por P, y de este modo se fija el posible valor de s en $g^s h^y \bmod p$. Obtenemos así **vinculación incondicional**.

A cambio, como ahora el posible valor de y es único, un V con suficiente capacidad computacional podría resolver el logaritmo discreto de x_1, obtener y, y entonces despejar s de x_2, antes de que P realice la apertura. Sin embargo, ante un V limitado, y suponiendo que el problema del logaritmo discreto es *difícil*, obtenemos **secreto computacional**. □

Una ventaja de este esquema frente al anterior, es que P puede elegir el valor de h por su cuenta, sin que V desconfíe de si conoce o no su logaritmo discreto. En el caso anterior, si P calcula maliciosamente h como $h = g^t \bmod p$, hemos visto que puede abrir un blob con el valor que quiera. En este esquema, gracias a la vinculación incondicional, aunque P conozca t, no puede romper el esquema de compromiso.

Para finalizar este capítulo, veamos las implicaciones de estos dos tipos de esquemas de compromiso existentes, los de secreto o vinculación incondicional, sobre las pruebas de conocimiento cero.

Cuando tratamos con un esquema de compromiso de vinculación incondicional, podemos trabajar incluso con un P de capacidad computacional ilimitada, como en las pruebas de conocimiento cero perfectas, y V estará seguro de que no se le engaña al abrir el testigo presentado. Sin embargo, desde la perspectiva de P, los blobs que no se abren, y deben ocultar la información, pueden verse comprometidos en el futuro si los problemas en los que se basa la propiedad de secreto computacional consiguen resolverse, bien para el caso general, o bien para las instancias de una cierta interacción pasada.

Por otro lado, si utilizamos esquemas de compromiso con secreto incondicional, P puede estar seguro de que los valores no revelados se mantendrán seguros en el futuro. Sin embargo, cuando un V se enfrenta a un P *todopoderoso*, la confianza en que P no cambiará el valor comprometido desaparece. Por eso, este tipo de compromisos sólo se pueden utilizar cuando en la prueba interactiva estudiada P no precisa de ser computacionalmente ilimitado.

5.5 PRUEBAS DE CONOCIMIENTO CERO COMPUTACIONALES

Introducimos ahora la última variedad de pruebas de conocimiento cero que veremos en este capítulo. La definición parte de la de conocimiento cero perfecta, pero considerando que los ensambles de la Vista y Simulador son indistinguibles sólo de manera computacional, sin necesidad de ser idénticos.

Definición 5.39 (Propiedad de conocimiento cero computacional)**.**
Un sistema de prueba interactiva (completo y robusto), para un problema de decisión Q, es *de conocimiento cero computacional* si el ensamble $\mathtt{Vista}_{P,V^*}(q,h)$ es indistinguible polinomialmente al ensamble generado por un Simulador $S_{V^*}(q,h)$, para cualquier instancia $\mathtt{Verdadera}\ q \in Q$ y cualquier historial h.

Para las pruebas que veremos en esta sección utilizaremos esquemas de compromiso con vinculación incondicional y secreto computacional, pues ante un P *todopoderoso*, debemos asegurarnos que no puede cambiar su compromiso. El esquema utilizado en cada algoritmo es arbitrario mientras cumpla la propiedad, de modo que denotaremos a un blob como una *caja negra* C que oculta un cierto valor, y lo desvela entregando su *llave* a V.

En esta clase de pruebas de conocimiento cero se encuentran problemas como el del camino hamiltoniano (HC), o el de la 3-coloración del grafo (G3C).

5.5.1 *Prueba de conocimiento cero para un grafo hamiltoniano*

Dado el problema de decisión del grafo hamiltoniano (Sección 3.3), mostramos el algoritmo descrito por Blum en [3]:

Algoritmo 5.40 (Prueba interactiva para HC)**.**
Datos comunes: Una instancia $G = (V, E)$ del Problema HC. $|\ V\ | = n$ es el tamaño del problema.
Protocolo:
P encuentra un ciclo hamiltoniano de G. Usará el mismo ciclo durante el resto de la prueba.
Sea $t(n)$ un polinomio en n. P y V repiten $t(n)$ veces los siguientes pasos.

1. P oculta G en cajas negras: asocia, de manera uniformemente aleatoria, los vértices $v_1, v_2, \ldots v_n$ a las cajas $C_1, C_2, \ldots C_n$ (una permutación sobre los vértices del grafo); además, para cada par de cajas (C_i, C_j) prepara otra caja llamada C_{ij} que guardará un 1 si los vértices ocultos en C_i y C_j son adyacentes, ó 0 en caso contrario.
2. $P \to V$: $\quad n + \binom{n}{2}$ cajas negras: $C_1, C_2, \ldots C_n, C_{1,2}, \ldots, C_{n-1,n}$
3. $V \to P$: $\quad b \in_R \{0, 1\}$.
4. Si $b = 0$, P envía las llaves para abrir todas las cajas.

   Si $b = 1$, P abre exactamente n cajas, $C_{ij}, C_{jk}, C_{kl}, \ldots C_{l'i}$, que corresponden al ciclo hamiltoniano escondido por los vértices escondidos en las cajas $C_1, C_2, \ldots C_n$.

5. Si $b = 0$, V comprueba que el grafo descubierto por las cajas sea G.

   Si $b = 1$, V comprueba que los índices de las cajas $C_{ij}, C_{jk}, \dots C_{l'i}$ forman un ciclo y que todas contienen un 1.

   Si la comprobación falla, V termina en rechazo. En caso contrario, vuelve al paso 1.

Tras $t(n)$ rondas, V termina y acepta la instancia como $\mathtt{Verdadera}$, G es hamiltoniano.

Teorema 5.41. *El protocolo 5.40 es una prueba interactiva.*

Demostración. El protocolo se ejecuta en $t(n)$ iteraciones, de modo que es probabilístico en tiempo polinomial. Ahora debemos comprobar la completitud y robustez.

El protocolo es *completo*, pues si el grafo es en verdad hamiltoniano, una instancia $\mathtt{Verdadera}$, P todopoderoso computacionalmente podrá encontrar un ciclo, y siguiendo el algoritmo, siempre podrá descubrir ante V el grafo G o el ciclo elegido, según el bit del reto.

Si un P^* tramposo no conoce un ciclo hamiltoniano de G, puede intentar adivinar cuándo V pedirá descubrir el grafo o el ciclo. En el primer caso, seguirá el protocolo, ocultando G en las cajas negras. En el segundo caso, P^* sólo debe elegir un ciclo de cajas $C_{ij}, C_{jk}, \dots C_{l'i}$ donde introducirá el valor 1, haya o no una arista uniendo los vértices en G. Si V pide descubrir el ciclo, P^* revelará un ciclo hamiltoniano, pero no del grafo G.

Si P^* no acierta correctamente qué bit enviará V, en cada ronda hay una probabilidad de $1/2$ de que V pille a P^* en su mentira. Ejecutando el protocolo $t(n)$ veces, la probabilidad de que un P^* tramposo engañe a V es de $2^{-t(n)}$, por tanto, la prueba es *robusta*. □

Teorema 5.42. *La prueba interactiva 5.40 es de conocimiento cero computacional.*

Demostración. Debemos construir un simulador para el protocolo anterior. Como en las pruebas anteriores, suponemos que V^* utiliza una función probabilística en tiempo polinomial F que con la información disponible elige el bit b del reto de manera no uniforme.

Simulador para el problema HC $S_{V^*}(G = (V, E), h)$.

Datos: G, una instancia $\mathtt{Verdadera}$ del problema HC; h, transcripciones de ejecuciones previas del protocolo; v_i, transcripción de la interacción actual (i rondas).

Ejecución: Repetir para $i + 1 \leqslant t(n)$:

1. Elegir $b_{i+1} \in_R \{0, 1\}$

2. **Si** $b_{i+1} = 0$, **entonces** oculta G en una permutación aleatoria de cajas negras.
   Si no, oculta un n-ciclo cualquiera en cajas negras.
   En ambos casos produce las cajas negras $C_1^{i+1}, \dots, C_{n-1,n}^{i+1}$.

3. **Si** $b_{i+1} = F(G, h, v_i, C_1^{i+1}, \dots, C_{n-1,n}^{i+1})$, **entonces** añadir la tupla $(C_1^{i+1}, \dots, C_{n-1,n}^{i+1}, b_{i+1}, K_{i+1})$ a la transcripción, donde K_{i+1} es el conjunto de claves para abrir las cajas, según el valor b_{i+1}.
   Si no, volver al paso 1.

4. $i = i + 1$

Al haber una probabilidad de $1/2$ de que el bit b_{i+1} coincida con el valor de la función F, el simulador es probabilístico, con un tiempo estimado de ejecución de 2 iteraciones por cada una de las $t(n)$ rondas, de modo que es de tiempo polinomial.

Tenemos los ensambles probabilísticos

$$\text{Vista}_{P,V^*}(G,h) = (G, h, E_1, B_1, K_1, \ldots, E_{t(n)}, B_{t(n)}, K_{t(n)}),\ y$$

$$S_{V^*}(G,h) = (G, h, E'_1, B'_1, K'_1, \ldots, E'_{t(n)}, B'_{t(n)}, K'_{t(n)}).$$

donde E_i y E'_i son las variables aleatorias que representan las cajas negras del compromiso, B_i y B'_i las v.a. de los bits elegidos por V en el reto, y K_i y K'_i las v.a. de las llaves reveladas.

Por inducción sobre i, cuando $i = 0$, los ensambles corresponden a la misma tupla (G, h), de modo que son idénticos.

Continuamos la inducción suponiendo que para $i - 1$ los ensambles son indistinguibles computacionalmente.

Analizando la prueba interactiva 5.40, en la interacción i, cuando V elige $b_i = 0$, éste obtiene de P una de las $n!$ posibles permutaciones del grafo G, y si V eligiera $b_i = 1$, obtendría un n-ciclo aleatorio, de los $n!$ posibles también, pues P forma dicho n-ciclo de cajas negras a partir de un ciclo hamiltoniano prefijado y la permutación aleatoria aplicada a G. La probabilidad de descubrir todo G o un n-ciclo depende de la elección de b_i, cuya probabilidad viene determinada por F.

Del mismo modo, analizando el simulador, F determinará la probabilidad de que en la transcripción aparezca un 0 o un 1 en el bit b_i. Según el valor de b_i, el simulador escogerá una de las $n!$ permutaciones posibles de G para ocultar el grafo en cajas negras, o bien uno de los $n!$ posibles n-ciclos, forme o no un ciclo hamiltoniano en G.

Ahora bien, suponíamos que las cajas negras eran en realidad la aplicación de un esquema de compromiso con vinculación incondicional, para proteger a V frente a un P *todopoderoso* que pudiese cambiar los valores comprometidos. Vimos en la sección anterior que al tener vinculación incondicional, sólo podemos aspirar a secreto computacional, en nuestro modelo de comunicación donde P y V ven todo lo que se les envía. Esto significa que la única manera de diferenciar los ensambles anteriores es rompiendo el secreto del esquema de compromiso, y suponemos que no existe ningún algoritmo probabilístico polinomial que pueda hacerlo.

Tenemos entonces, que los ensambles son computacionalmente indistinguibles, y la prueba interactiva es de conocimiento cero computacional. □

5.5.2 *Prueba de conocimiento cero para la 3-coloración de un grafo*

Otra prueba interactiva de conocimiento cero computacional es la de demostrar que un grafo G posee una 3-coloración de sus nodos (Sección 3.4).

Como los problemas G3C y HC, de las sección anterior, son **NP**-completos, y como todos los problemas **NPC** son computacionalmente equivalentes, en el sentido de que todos se pueden reducir polinomialmente a otro problema **NPC**, tenemos que todo problema en **NPC** tiene una prueba de conocimiento cero computacional.

Blum presenta en [3] una prueba de conocimiento cero para G3C, que sigue las líneas de la prueba del ciclo hamiltoniano, utilizando un bit como reto para desvelar el grafo

G o una prueba de que se ocultó una 3-coloración. Sin embargo, en [16] se muestra una variación de la prueba que creemos es más sencilla de entender.

La idea de esta prueba es que en cada iteración, P elige una permutación aleatoria de la 3-coloración, $\pi \in \mathrm{Sym}(\{1,2,3\})$, y oculta cada vértice coloreado en una caja negra. El *testigo* que enviará a V consistirá en ocultar toda la 3-coloración del grafo en n cajas negras cuyos índices corresponden a los vértices del grafo. V elegirá como *reto* una arista aleatoria. Como *respuesta*, P enviará la coloración permutada de esa arista, que deberá tener colores distintos en cada extremo, y las claves de cifrado de cada una, de modo que V podrá comprobar si corresponden al *testigo* enviado:

Algoritmo 5.43 (Prueba interactiva para G3C)**.**

Datos comunes: Una instancia $G = (V, E)$ del Problema G3C. $|\ E\ | = m$ y $|\ V\ | = n$ es el tamaño del problema.

Protocolo:

P encuentra una 3-coloración ϕ de G.

Sea $t(n)$ un polinomio en n. P y V repiten $t(n)\ |\ E\ |$ veces los siguientes pasos.

1. P selecciona al azar una permutación $\pi \in_R \mathrm{Sym}(\{1,2,3\})$. Oculta en n cajas negras la coloración: la caja C_j ocultará el valor $\pi(\phi(v_j))$ del color permutado del vértice v_j.
2. $P \to V$: n cajas negras: $C_1, C_2, \ldots C_n$.
3. $V \to P$: $(u, v) \in_R E$, una arista aleatoria de G.
4. $P \to V$: las claves k_u, k_v que abren las cajas C_u, C_v.
5. V abre las cajas y comprueba que $\pi(\phi(u)) \neq \pi(\phi(v))$.

   Si la comprobación falla, V termina en rechazo. En caso contrario, vuelve al paso 1.

Tras $t(n)$ rondas, V termina y acepta la instancia como `Verdadera`, G posee una 3-coloración.

Teorema 5.44. *El protocolo 5.43 es una prueba interactiva.*

Demostración. La completitud es inmediata, pues si existe 3-coloración, un P ilimatado computacionalmente la podrá calcular, y podrá dar para cada $(u, v) \in E$ colores distintos en cada ronda.

Si P^* no conoce la 3-coloración, al menos hay una arista a la que P no puede dar colores distintos a sus vértices, y hay una probabilidad de $\frac{1}{|E|}$ de que V le rechace, es decir, una probabilidad $1 - \frac{1}{|E|}$ de que P^* triunfe. Tras $t(n)\ |\ E\ |$ rondas, P^* sólo engañaría a V con una probabilidad $(1 - \frac{1}{|E|})^{t(n)|E|} \approx e^{-t(n)}$, por lo que la prueba es robusta.

□

Teorema 5.45. *La prueba interactiva 5.43 es de conocimiento cero computacional.*

Demostración. La propiedad de conocimiento cero se prueba con un simulador que funciona como los anteriores, primero se elige el *reto* (arista) y la *respuesta* (dos colores), se calcula el *testigo* respecto de ambos y se comprueba si V hubiera elegido dicho *reto* a partir de un algoritmo probabilístico polinomial F:

Simulador para el problema G3C $S_{V^*}(G = (V, E), h)$.

Datos: G, una instancia $Verdadera$ del problema G3C; h, transcripciones de ejecuciones previas del protocolo; v_{i-1}, transcripción de la interacción actual ($i-1$ rondas).
Ejecución: Repetir para $i+1 \leqslant t(n)$:

1. Elegir $e = (u, v) \in_R E$ una arista, y sus colores $(a, b) \in_R \{(\alpha, \beta) \mid \alpha \neq \beta;\ \alpha, \beta \in \{1, 2, 3\}\}$ de manera aleatoria.

2. Generar las cajas negras C_j para $j = 1, \ldots n$ del siguiente modo:

$$C_j \text{ oculta} \begin{cases} a & \text{si } v_j = u, \\ b & \text{si } v_j = v, \\ 0 & \text{en otro caso.} \end{cases}$$

   donde v_j es un vértice de G.

3. **Si** $e = F(G, h, v_{i-1}, C_1, \ldots, C_n)$, **entonces** añadir la tupla $(C_1, \ldots, C_n, e = (u, v), k_u, k_v)$ a la transcripción, donde K_{i+1} es el conjunto de claves para abrir las cajas, según el valor b_{i+1}.
   Si no, volver al paso 1.

4. $i = i + 1$

Durante una ronda del simulador tenemos una probabilidad de $\frac{1}{|E|}$ de que la arista escogida en el primer paso coincida con la que el algoritmo F elegiría, con $t(n)$ rondas a generar. Tenemos un algoritmo probabilístico con tiempo esperado de ejecución polinomial.
Veamos ahora los ensambles de la Vista y la Simulación.

$$Vista_{P,V^*}(G, h) = (G, h, C_1, E_1, K_1, \ldots, C_{t(n)}, E_{t(n)}, K_{t(n)}),\ y$$

$$S_{V^*}(G, h) = (G, h, C'_1, E'_1, K'_1, \ldots, C'_{t(n)}, E'_{t(n)}, K'_{t(n)}).$$

Donde la v.a. C_i representa el conjunto de n cajas negras que oculta la permutación de la coloración, la v.a. E_i representa la arista (u, v) elegida como reto por V en la ronda i, y la v.a. K_i representa el par de llaves (k_u, k_v) enviado como respuesta.
Por inducción sobre i, cuando $i = 0$, los ensambles de la Vista y Simulador son idénticos, (G, h). Suponemos cierto para $i-1$, en la i-ésima iteración tendremos:
En la Vista y el Simulador, las variables aleatorias C_i y C'_i corresponden a n cajas negras, que cumplen un esquema de compromiso con vinculación incondicional y secreto computacional, de modo que a partir de estas n cajas negras suponemos que no existe ningún

algoritmo probabilístico en tiempo polinomial que las distinga de cualquier otra caja negra. Por tanto, la probabilidad de que aparezcan unas cajas negras u otras en la Vista y Simulador es computacionalmente indistinguible e independiente de las otras variables.

Respecto a las v.a. E_i y E'_i, ambas dependen de la probabilidad de elegir un nodo de F, que podemos supones es p_F, idéntica para ambas.

En la apertura de las cajas negras con las claves de K_i y K'_i se desvelan dos colores distintos aleatorios, en la Vista porque la 3-coloración implica que ambos son distintos y después se le aplica una permutación aleatoria de entre las 3! posibles, por lo que dos colores distintos tienen una probabilidad uniforme de ser revelados, y por tanto, la misma probabilidad que la elección de colores aleatoria del Simulador.

Tal como ocurría en la prueba del grafo hamiltoniano, la condición de conocimiento cero computacional proviene de utilizar un esquema de compromiso con vinculación incondicional, y secreto computacional. □

6

APLICACIONES DE LAS PRUEBAS DE CONOCIMIENTO CERO

En este capítulo nos alejamos del estudio teórico de las pruebas de conocimiento cero para introducirnos en sus aplicaciones prácticas. La primera técnica que veremos consiste en transformar una prueba interactiva en no interactiva, aprovechando para generar un esquema de firma digital basado en pruebas de conocimiento cero. A continuación, veremos cómo se aplican las pruebas de conocimiento cero en un esquema de identificación de usuarios. Finalmente, como evolución de esos esquemas de identificación, estudiaremos cómo funcionan los certificados digitales de Identity Mixer diseñados por IBM y basados en pruebas de conocimiento cero. Se puede consultar la bibliografía de este capítulo en [14, 17, 19].

6.1 PRUEBAS DE CONOCIMIENTO CERO NO INTERACTIVAS

Hemos visto que la interacción habitual en una prueba de conocimiento cero consiste en enviar un *testigo* u, un *reto* b y una *respuesta* $w = \xi(u, b)$ en cada iteración, para que V verifique que el testigo corresponde al reto y respuesta enviados, $u = \vartheta(b, w)$.

La desventaja de estos protocolos en la práctica es sincronizar a P y V para comunicarse, por la necesidad de que el Verificador debe generar el reto él mismo, para estar seguro de que la prueba es correcta (por la propiedad de conocimiento cero). Para solucionarlo, querríamos generar pruebas de conocimiento cero de manera asíncrona, con un solo mensaje a ser verificado cuando V lo reciba.

Una función de *hash*, o resumen digital, recibe un mensaje arbitrario como entrada, y tras una serie de operaciones sobre el mismo, devuelve una cantidad fija de bits que es difícil de anticipar antes de ejecutar la función. Una propiedad de estas funciones es que con un mínimo cambio en el mensaje original, el resultado devuelto cambia drásticamente.

Una técnica para convertir las pruebas de conocimiento cero en no interactivas es la *heurística de Fiat-Shamir*, la cual consiste en sustituir el reto b por un resumen digital del testigo u, de modo que para un P computacionalmente limitado, éste no puede anticipar el valor del hash, y así V confía en que P no ha elegido el u adecuado para falsear la prueba.

Aprovechando la heurística de Fiat-Shamir, podemos convertir el protocolo en un esquema de firma digital de un mensaje m, que equivale a una firma real sobre un documento de papel que sería el mensaje. Aplicando el resumen digital sobre el testigo y el mensaje a la vez, asociamos el mensaje a la prueba de conocimiento cero no interactiva:

P calcula :	el *testigo* u, el *reto* $h =$**hash(u | m)**, y la *respuesta* $w = \xi(u, h)$.
P $\to$ V :	***firma del mensaje*** m: (h, w)
V verifica :	$h = \mathrm{hash}(\,\vartheta(h, w) \,|\, m\,)$.

Observación.

En las pruebas de conocimiento cero perfectas estudiadas, los retos consistían en un bit, por lo que los posibles valores son 0 ó 1 y un ataque a la función de *hash* es muy fácil.

Por ello, se utilizan pruebas de conocimiento cero como el protocolo de Schnorr, donde el conjunto de los posibles valores del reto es suficientemente grande, usualmente unas centenas de bits.

6.2 PROTOCOLOS DE IDENTIFICACIÓN BASADOS EN PRUEBAS DE CONOCIMIENTO CERO

Las pruebas de conocimiento cero tienen una gran aplicación en el campo de la seguridad informática, en particular en la **autenticación** de usuarios, por ejemplo, al acceder al servidor de correo electrónico, o al aula virtual. Un sistema de identificación basado en pruebas de conocimiento cero consigue por un lado privacidad, al no revelar información del usuario, y por otro lado seguridad, al no *degradarse* con el uso, es decir, resiste al criptoanálisis por muchos mensajes que se intercepten.

6.2.1 *Protocolo de identificación de Fiat-Shamir*

El protocolo de identificación de Fiat-Shamir, basado en la prueba de conocimiento cero del problema QR, es el más característico junto con el de Schnorr (Sección 5.3).

Como nuestro usuario P no es computacionalmente todopoderoso, partiremos de que P conoce s, la raíz modular de un residuo cuadrático v, lo cual le permitiría realizar la prueba con éxito sin conocer la factorización de N. V autenticará al usuario P si éste es capaz de demostrar que v es un residuo cuadrático, sin revelar s.

Algoritmo 6.1 (Protocolo de identificación Fiat-Shamir)**.**
Configuración de la identidad:

1. La entidad de confianza selecciona y publica $N = pq$, con p y q primos y secretos.
2. Cada usuario P genera un secreto $s \in \mathbb{Z}_N^*$, coprimo con N (si no, se podría obtener la factorización de N y perder la seguridad del protocolo). Calcula $v \equiv s^2 \bmod N$ y lo envía a la entidad de confianza como su clave pública.

Protocolo: Repetir t rondas:

1. P escoge aleatoriamente $r \in_R \mathbb{Z}_N^*$, el *compromiso*.
2. $P \to V$: $u \equiv r^2 \bmod N$, el *testigo*.
3. $V \to P$: $b \in_R \{0, 1\}$, el *reto*.
4. $P \to V$: $w \equiv r \cdot s^b \bmod N$, la *respuesta*.
5. V verifica si $w^2 \equiv u \cdot v^b \bmod N$.

El protocolo de Fiat-Shamir es casi idéntico a la prueba interactiva 5.4, donde aquí (v, N) hace el papel de instancia $\mathtt{Verdadera}$ del problema QR, y como P, el usuario, es una máquina computacionalmente limitada, en vez de elegir aleatoriamente el residuo cuadrático u y la raíz cuadrada w, parte de las raíces modulares s y r para poder calcular u, v y w residuos cuadráticos.

El intento de ataque que vimos en la demostración de robustez varía ligeramente para Fiat-Shamir, pues el objetivo aquí es intentar demostrar que v es residuo cuadrático, sin conocer s u otra raíz cuadrada, mientras que en la prueba interactiva del capítulo anterior, el objetivo era demostrar que un no-residuo cuadrático era un residuo cuadrático.

Un atacante debe adivinar el bit del reto que recibirá, de modo que si cree que $b = 0$ calcula el testigo como en el protocolo, pero si cree que $b = 1$, calcula en el paso 2 el testigo $u \equiv r^2 \cdot v^{-1} \bmod N$, y en 4 la respuesta $w \equiv r \bmod N$. En ambos casos fallaría si no adivina b correctamente, pues para corregir su error debería ser capaz de calcular, respectivamente, una raíz cuadrada de v, que permitiría pasar la prueba siempre, o una raíz cuadrada de $u \equiv r^2 \cdot v^{-1} \bmod N$, computacionalmente inviable pues p y q los guarda como secretos la entidad de verificación.

Como en la prueba interactiva, un atacante tiene una probabilidad de $1/2$ de engañar a V en cada ronda, de modo que la robustez se mantiene con una probabilidad de ataque de 2^{-t}. Si P pasa correctamente las t rondas, V da por válida la prueba. Si falla aunque sea sólo una, rechaza la identificación.

6.3 PRUEBAS DE CONOCIMIENTO CERO EN IDENTITY MIXER

Identity Mixer, o Idemix para acortar, es un protocolo desarrollado por IBM[1] para crear certificados digitales basados en atributos, donde se preserva la privacidad. Este protocolo no solo permite la autenticación sin revelar un valor secreto, también permite realizar pruebas sobre los atributos del certificado de un usuario, sin revelar los valores, como por ejemplo, "`año_actual - año_nacimiento` $\geqslant$ `18`".

Los protocolos criptográficos en los que se basa Idemix [17] han sido desarrollados durante años por expertos en el área, donde destacan Jan Camenisch y Anna Lysyanskaya con el desarrollo de la firma CL [5] [4] que se utiliza para firmar certificados sin tener que conocer los valores de todos los atributos, y que permite posteriormente al usuario demostrar que posee dicha firma, sin desvelarla. Esto permite utilizar múltiples veces un certificado de manera anónima sin que se relacionen los diferentes usos con la misma persona.

6.3.1 *Notación para ZKP*

Como hemos visto, cualquier problema de decisión posee una prueba de conocimiento cero, y de entre esas pruebas, las pruebas de conocimiento de un secreto son las que se utilizan en los certificados. Utilizaremos la siguiente notación de Camenisch y Stadler [6] para denotar una prueba de conocimiento cero de conocimiento de un secreto.

$$\mathsf{ZKPoK}\{(w) : \mathcal{L}(w, x)\}$$

donde w es un valor secreto, x es información conocida por ambas partes, y $\mathcal{L}(w, x)$ es un predicado que representa una condición sobre w y x. Se puede leer como "conozco un secreto w tal que el predicado $\mathcal{L}(w, x)$ se cumple para w y x". Para abreviar, en vez de ZKPoK (*Zero-Knowledge Proof of Knowledge*), usaremos ZKP o PK.

Por ejemplo, $\mathsf{PK}\{(\alpha) : y = g^{\alpha}\}$ donde $y \in G =< g >$, un grupo de orden q primo, denota el protocolo de identificación de Schnorr.

1 Identity Mixer - https://www.research.ibm.com/labs/zurich/idemix/

6.3.2 *Probar el conocimiento de una representación*

Una generalización de Schnorr utilizada en Idemix es utilizar l bases $g_1, \ldots, g_l$ con $g_i \in G$, siendo g un generador de G con orden $\mathrm{ord}(g) = q$ primo. Sea $y \in G$ tal que conocemos los valores $x_1, \ldots, x_l$ que cumplen $y = \prod_{i=1}^{l} g_i^{x_i}$. Una prueba de conocimiento cero de que conocemos la representación de y con respecto a las bases $g_1, \ldots, g_l$ consiste en:

Algoritmo 6.2.

P y V conocen y, $g_1, \ldots, g_l$, q y el parámetro del sistema k.

1. P escoge aleatoriamente l enteros $r_i \in_R \mathbb{Z}$.
2. P $\to$ V: $\quad t = \prod_{i=1}^{l} g_i^{r_i}$.
3. V $\to$ P: $\quad c \in_R \{0,1\}^k$, un entero aleatorio de k bits.
4. P $\to$ V: $\quad s_i = r_i - c \cdot x_i \bmod q$, para $i = 1, \ldots, l$.
5. V verifica si $\quad t = y^c \prod_{i=1}^{l} g_i^{s_i}$.

La probabilidad de que un P malicioso adivine los k bits aleatorios del reto c es de 2^{-k}. Comprobamos que el protocolo funciona pues

$$y^c \prod_{i=1}^{l} g_i^{s_i} = \left(\prod_{i=1}^{l} g_i^{x_i} \right)^c \prod_{i=1}^{l} g_i^{s_i} = \prod_{i=1}^{l} g_i^{cx_i + r_i - cx_i} = \prod_{i=1}^{l} g_i^{r_i} = t$$

6.3.3 *Firma Camenisch-Lysyanskaya*

El esquema de firma de Camenisch-Lysyanskaya, o firma CL, es la base de los certificados Idemix. Procedemos a describirlo.

Sean p' y q' dos primos, tal que $p = 2p' + 1$ y $q = 2q' + 1$ también lo son, llamados *primos seguros*. Los parámetros del sistema, conocidos por el firmante y quien comprueba la firma son: el entero $n = pq$, $Z, S, R \in QR_n$ (grupo multiplicativo de los residuos cuadráticos módulo n). La clave secreta del firmante es (p, q) y el mensaje lo denominaremos m. La firma CL sobre m es la tupla (A, e, v) tal que $A \equiv \left(\frac{Z}{S^v R^m} \right)^{1/e} \bmod n$, donde e, v son aleatorios, e primo y $\frac{1}{e} \cdot e \equiv 1 \bmod \varphi(n)$.

Para verificar que una firma (A, e, v) es correcta, basta comprobar si $Z \stackrel{?}{\equiv} A^e S^v R^m \bmod n$.

La ventaja de esta firma es que se puede aplicar sobre múltiples mensajes a la vez, que en el caso de un certificado, serán los diferentes atributos a firmar. En este caso los parámetros del sistema se mantienen, $n = pq$, $Z, S \in QR_n$, pero debemos calcular un residuo cuadrático por cada mensaje a firmar, $R_0, \ldots, R_l \in QR_n$. La clave secreta sigue siendo (p, q), y la firma de los mensajes $m_0, m_1, \ldots, m_l$ es (A, e, v) tal que

$$A \equiv \left(\frac{Z}{S^v \prod_{i=0}^{l} R_i^{m_i}} \right)^{1/e} \bmod n$$

Para verificar la firma, comprobamos que $Z \stackrel{?}{\equiv} A^e S^v \prod_{i=0}^{l} R_i^{m_i} \bmod n$.

6.3.4 *Firma Camenisch-Lysyanskaya aleatorizada*

Dada una firma CL (A, e, v), sobre los mensajes $m_0, m_1, \ldots, m_l$, podemos obtener una firma $(\acute{A}, e, \hat{v})$ distinta de la anterior, excepto por e, que todavía sea válida. Esta nueva firma la puede generar cualquier usuario, pues no precisa del secreto (p, q) de la firma original.

La nueva firma se obtiene a partir de un entero aleatorio r, calculando $\acute{A} = A \cdot S^{-r} \bmod n$ y $\hat{v} = v + er$.

La verificación de $(\acute{A}, e, \hat{v})$ se realiza igual que en la firma original, ya que:

$$\acute{A}^e S^{\hat{v}} \prod_{i=0}^{l} R_i^{m_i} \equiv A^e S^{-er} S^v S^{er} \prod_{i=0}^{l} R_i^{m_i} \equiv A^e S^v \prod_{i=0}^{l} R_i^{m_i} \equiv Z \bmod n$$

6.3.5 *Firma de credenciales Idemix*

Una credencial Idemix consiste en una lista de atributos con valores, la cual estará firmada por un Emisor de confianza, como puede ser un gobierno al firmar el DNI electrónico. Sin embargo, la firma CL del certificado Idemix la calcularán interactivamente el Usuario y el Emisor, pues ninguno tendrá toda la información necesaria para generar la firma por sí mismo. En cada paso, una prueba de conocimiento cero asegurará que cada parte ha seguido el protocolo.

El Emisor conocerá la factorización de n, mientras que el Usuario guardará una clave secreta, m_0 y, opcionalmente, los atributos del certificado, $m_1, \ldots, m_l$. Al final de la firma, el Usuario conocerá una representación, $(m_0, \ldots, m_l)$ de $\frac{Z}{A^e S^v}$ con respecto a la base $(R_0, \ldots, R_l)$, y como ya hemos visto, podrá demostrar con una prueba de conocimiento cero que conoce dicha representación sin revelar necesariamente ninguno. Además, el Emisor no conocerá la firma (A, e, v) final, pues v se calculará también de manera conjunta de modo que sólo el Usuario conocerá la firma generada.

A continuación, presentamos una versión simplificada del protocolo, donde todos los atributos, menos la clave secreta, son conocidos por el Emisor. En otras versiones el Usuario puede enviar pruebas sobre alguna propiedad del atributo sin revelar el verdadero valor al Emisor, pero esto depende de la política del Emisor al firmar certificados. Es importante notar que el número de residuos cuadráticos generados por el emisor, $R_0, \ldots, R_{l'}$ determina el máximo de atributos que puede llevar un certificado.

Algoritmo 6.3 (Emisión de certificado Idemix)**.**
Conocimiento común: $n, Z, S, R_0, \ldots, R_{l'}, m_1, \ldots, m_l$, donde $l' \geqslant l$
Conocimiento del Emisor: p, q tal que $n = pq$.
Conocimiento del Usuario: m_0, su clave secreta.

Protocolo:

1. Ronda 1: Emisor
   1.1. El Emisor escoge un valor aleatorio n_1 y lo envía al Usuario.
2. Ronda 2: Usuario
   2.1. El Usuario elige un valor v' aleatorio (primer paso para calcular el v de la firma).

2.2. Calcula el valor $U = S^{v'} \cdot R_0^{m_0} \bmod n$.

2.3. Calcula* la prueba $P_1 := ZKP\{(v', \mu_0) : U \equiv S^{v'} R_0^{\mu_0} \bmod n\}(n_1)$, no interactiva, dependiente de n_1.

2.4. Envía (U, P_1, n_2), donde n_2 es un valor aleatorio elegido por el Usuario.

3. Ronda 3: Emisor firma CL

3.1. Verifica* la prueba P_1 del Usuario.

3.2. Elige un valor un e primo y v'' aleatorios (segundo paso para calcular el v de la firma).

3.3. Calcula el valor $A = \left(\frac{Z}{U \cdot S^{v''} \cdot \prod_{i=1}^{l} R_i^{m_i}} \right)^{1/e} \bmod n$.

3.4. Calcula* $P_2 := ZKP\{(\delta) : A \equiv \left(\frac{Z}{U \cdot S^{v''} \cdot \prod_{i=1}^{l} R_i^{m_i}} \right)^{\delta} \bmod n\}(n_2)$, no interactiva, dependiente de n_2.

3.5. Envía (A, e, v'') y P_2 al Usuario.

4. Ronda 4: Usuario

4.1. Calcula $v = v' + v''$.

4.2. Verifica P_2 y la firma CL (A, e, v).

---

*Las pruebas y verificaciones correspondientes pueden ser consultados en el Apéndice B.

6.3.6 *Revelación selectiva de atributos Idemix*

Tras la ejecución del siguiente protocolo, el Verificador conocerá algunos de los atributos del certificado, y una prueba de que el Emisor lo firmó, junto a unos atributos que no le mostramos, incluida la clave secreta m_0. El Verificador tampoco conocerá la firma CL original (A, e, v), sólo A' de una versión de la firma aleatorizada.

Partiendo de un certificado con la clave secreta m_0, los atributos $m_1, \ldots, m_l$ y la firma CL (A, e, v), vamos a revelar a un Verificador todos nuestros atributos excepto los dos primeros, m_1 y m_2, y por supuesto, tampoco la clave privada m_0.

Para esto, primero aleatorizaremos la firma CL como vimos en 6.3.4 y realizaremos una prueba no interactiva de que conocemos la representación (m_0, m_1, m_2) (6.3.2) de un cierto valor en la base R_0, R_1 y R_2:

Algoritmo 6.4 (Revelación selectiva).

Conocimiento común: $n, Z, S, R_0, \ldots, R_l, m_3, \ldots, m_l$.

Conocimiento del Usuario: m_0, su clave secreta, m_1 y m_2, sus atributos ocultos, (A, e, v) su firma CL.

Protocolo:

1. V$\rightarrow$ P : valor aleatorio n_1.

2. Usuario aleatoriza la firma CL:

2.1. r aleatorio, $(A' := AS^{-r} \bmod n,\ e,\ v' := v + er)$

3. Usuario calcula la prueba de conocimiento no interactiva:
   $PK\{(\hat{e}, \hat{v}, m_0, m_1, m_2) : \frac{Z}{\prod_3^l R_i^{m_i}} \equiv A'^{\hat{e}} S^{\hat{v}} \prod_{i=0}^{2} R_i^{m_i} \bmod n\}(n_1)$

   3.1. Elige valores aleatorio $\tilde{e}$ y $\tilde{v}'$.

   3.2. Elige valores aleatorios $\tilde{m}_0$, $\tilde{m}_1$ y $\tilde{m}_2$.

   3.3. Calcula $\tilde{Z} := (A')^{\tilde{e}} \left(\prod_{i=0}^{3} R_i^{\tilde{m}_i} \right) S^{\tilde{v}'} \bmod n$.

   3.4. Genera el reto por Fiat-Shamir $c := Hash(A' \| \tilde{Z} \| n_1)$.

   3.5. Calcula: $\hat{e} := \tilde{e} + ce;\ \hat{v}' := \tilde{v} + cv';\ \hat{m}_i := \tilde{m}_i + cm_i$, para $i = 0, 1, 2$.

   3.6. $P_1 := (c, \hat{e}, \hat{v}', \hat{m}_0, \hat{m}_1, \hat{m}_2)$.

4. $U \to V : A'$ y P_1.

5. Verificador comprueba:

   5.1. Calcula $\hat{Z} := \left(\frac{Z}{\prod_{i=3}^{l} R_i^{m_i}} \right)^{-c} (A')^{\hat{e}} \left(\prod_{i=0}^{3} R_i^{\hat{m}_i} \right) S^{\hat{v}'} \bmod n$.

   5.2. Acepta si $c = Hash(A' \| \hat{Z} \| n_1)$.

Un ejemplo cercano de uso de este protocolo podría ser las encuestas anónimas de asignaturas. Actualmente, si queremos asegurar al alumno que la encuesta es totalmente anónima, sin repercusiones, debemos dejarla *abierta*, pudiendo cualquier persona rellenarla. Si queremos asegurar que sólo los alumnos matriculados puedan acceder a la encuesta, tendrían que iniciar sesión con su correo electrónico, y que confíen en que nadie accederá a sus comentarios.

Con un certificado Idemix, un atributo podría ser "Alumno de la asignatura X", otro sus datos de identificación. La Universidad firmó su certificado, que puede llevar en la tarjeta inteligente, por lo que disponemos de una firma CL. Al iniciar sesión en la encuesta, bastaría con mostrar que se es alumno de la asignatura, y generar la prueba no interactiva de que posee un certificado firmado por la Universidad. Sólo los verdaderos alumnos podrán acceder, y ningún registro puede relacionar una opinión con su autor.

Otras soluciones criptográficas tradicionales sólo se preocupaban de proteger la transmisión, que el mensaje no pudiera ser interceptado. Con las pruebas de conocimiento cero hemos conseguido proteger los mismos datos, ante un interlocutor en quien no confiamos.

CONCLUSIONES

Como cierre de este trabajo podemos destacar los conocimientos obtenidos durante el camino. Éste ha sido un trabajo de investigación que permite tratar un área de las matemáticas muy relacionada con la informática, desde que en el primer capítulo tratásemos la teoría de la computación hasta las aplicaciones de seguridad al final de la memoria. Sin embargo, los problemas matemáticos analizados y usados, como base de la criptografía y pruebas de conocimiento cero, no es una teoría que se estudie al mismo nivel en el Grado en Matemáticas que en otros estudios. El estudio formal que hemos realizado para poder entenderlas es fundamentalmente matemático, pero con aplicación a otras áreas.

Durante el seminario *Grupos y criptografía: de Julio César a las curvas elípticas* de Adolfo Quirós, el pasado abril de 2017, se hizo el acertado comentario de que los problemas utilizados en criptografía no eran difíciles de entender. En efecto, hemos utilizado problemas de distintas áreas de las matemáticas, y los enunciados de los mismos no requerían un estudio extensivo sobre sus respectivas áreas para comprenderlos. Sin embargo, han demostrado que hasta el día de hoy, lo difícil es resolverlos. Por ejemplo, en el problema de la factorización vimos cómo las técnicas más avanzadas conseguían tiempos subexponenciales que sólo sirven en la práctica para factorizar números de tamaño relativamente pequeño.

También construimos unas herramientas fundamentales, los esquemas de compromiso, donde una de sus propiedades básicas, secreto o vinculación, dependían de la dificultad de resolver los problemas de teoría de números o grafos vistos.

Y también basándonos en esa dificultad, construimos algoritmos polinomiales que los pueden resuelven, a costa de suponer que una máquina P todopoderosa existe y es capaz de resolverlos, anulando en P la dificultad de los problemas. Modificando condiciones ajenas a P pasamos de pruebas perfectas, donde sabemos que de la prueba ninguna máquina podría obtener más información, a pruebas con Verificador Honesto, donde no conocemos ataques eficaces, o pruebas computacionales, donde la seguridad del conocimiento cero dependía de la dificultad del problema utilizado en el esquema de compromiso.

Sin embargo, si queremos utilizar dichos métodos en la práctica, nuestro P también estará limitado computacionalmente, y deberá conocer de antemano una solución al problema, sin tener que calcular la solución. Es lo que vimos en la variación de la prueba de conocimiento cero de los residuos cuadráticos en el algoritmo de identificación de Fiat-Shamir.

Finalmente, hemos visto que las aplicaciones de las pruebas de conocimiento cero no se quedan sólo en pequeñas interacciones para identificación, sino que permiten realizar tareas más complejas como computación distribuida, que se combinan en un protocolo como Idemix.

Inicialmente elegimos este área de trabajo debido a que para el TFG de Ing. Informática no había cabida para estudiar como se merecía la criptografía de Idemix, junto a su implementación en entornos IoT, tema central de dicho trabajo. Aprovechando que en el plan de estudios del doble grado ahora debíamos realizar dos Trabajos de Fin de Grado, separamos la teoría y la práctica de modo que no hubiera ningún solapamiento entre los trabajos presentados. Sin embargo, ambos se complementaron indirectamente, pues desarrollar uno ayudaba a comprender mejor el otro. En mi opinión personal, creo que fue una buena decisión que ahora refleja las ventajas del doble grado.

ANEXO

A

IMPLEMENTACIONES

En este capítulo mostramos las implementaciones de algunos algoritmos descritos durante el trabajo. Utilizamos para esto un *sistema algebraico computacional* (CAS), de código abierto, llamado `Sage`, construido sobre conocidos paquetes matemáticos como `NumPy`, `Sympy`, `R`, `PARI/GP` o `Maxima`, y que utiliza para programar un lenguaje basado en `Python`.

La ventaja de esta herramienta es la facilidad para manejar estructuras de datos y operaciones que en otros lenguajes sería muy largo de implementar, lo que facilita, además, la lectura del código.

A.1 PRUEBA INTERACTIVA: PROBLEMA DEL RESIDUO CUADRÁTICO

En el capítulo de Pruebas de Conocimiento Cero, la primera prueba interactiva que vimos fue aquella basada en el problema del residuo cuadrático, donde se demuestra que un elemento x es un cuadrado módulo N. Aquí mostramos dos versiones, una donde P conoce una raíz cuadrada de x, y otra versión donde conoce la factorización de N, pues ya vimos que el problema QR es equivalente al problema de la factorización.

Listing 1: ZKP QR almacenando la raíz modular

```
s = 0
t = 0

def Setup():
  global s
  p = random_prime(2^100)
  q = random_prime(2^100)
  N = p*q
  s = Zmod(N).random_element()
  x = s^2
  return x,N

def elegir_u(x,N):
  global t
  t = Zmod(N).random_element()
  return t^2

def enviar_prueba(x,N,u,b):
  if b == 0:
    return t
  else:
    return Zmod(N)(t*s)

x,N = Setup()
u = elegir_u(x,N)
w = enviar_prueba(x,N,u,1)

print "Comprobar"
```

```
print Zmod(N)(w^2)
print Zmod(N)(u*x)
```

Durante la etapa de `Setup` se eligen dos primos aleatorios, p y q, con una magnitud de hasta 2^{100}. El módulo del problema QR será $N = pq$, un número compuesto donde resolver una instancia del problema es difícil. Por último, se elige un valor aleatorio s (de **s**ecreto) en $\mathbb{Z}_N$, cuyo cuadrado será el residuo cuadrático x de la instancia actual del problema. Los valores de p, q y s sólo los conocerá P, mientras que la información común a P y V es N y x, la instancia del problema QR.

A continuación, siguiendo el protocolo 5.4 , P elige su *compromiso* u, un residuo cuadrático de $\mathbb{Z}_N$ al azar. Lo calculamos como x antes, eligiendo un t al azar, y elevando al cuadrado. El valor de t será una de sus raíces cuadradas módulo N.

Por último, `enviar_prueba` recibe como parámetro el *reto* b de V. Según el protocolo, P debe devolver una raíz cuadrada módulo N de u o de $x \cdot u$.

En Listing 1, gracias a que P guarda los valores de s y t, raíces de x y u, respectivamente, le basta con enviar t o $s \cdot t \bmod N$.

En cambio, como vimos en el capítulo de los residuos cuadráticos, la raíz módulo un número primo es fácil de calcular. Por eso, si conocemos la factorización de N, podemos utilizar el Teorema Chino de los Restos para calcular una raíz de $u \cdot x^b \bmod N$ a partir de las raíces módulo p y q. La versión de Listing 2 aplica esta idea, utilizando herramientas de `Sage`, cómo la raíz modular, `ZZ(sqrt(zp))`, o el Teorema Chino de los Restos, `crt()` (Chinese Remainder Theorem).

Listing 2: ZKP QR utilizando TCR

```
p = 0
q = 0

def Setup():
  global p,q
  p = random_prime(2^100)
  q = random_prime(2^100)
  N = p*q
  s = Zmod(N).random_element()
  x = s^2
  return x,N

def elegir_u(x,N):
  t = Zmod(N).random_element()
  return t^2

def enviar_prueba(x,N,u,b):
  if b == 0:
    z = u
  else:
    z = x*u
  zp = Zmod(p)(z)
  zq = Zmod(q)(z)
  w = crt([ZZ(sqrt(zp)),ZZ(sqrt(zq))],[p,q])
  return w

x,N = Setup()
```

```
u = elegir_u(x,N)
w = enviar_prueba(x,N,u,1)

print "Comprobar"
print Zmod(N)(w^2)
print Zmod(N)(u*x)
```

La optimización de operaciones utilizando la factorización de N y el TCR es una técnica utilizada en otros protocolos criptográficos conocidos, como RSA [18], pero el hecho de almacenar p y q, además de la clave privada s, obliga a tener que aplicar medidas de seguridad a más de un valor, existiendo más puntos de ataque a un usuario.

A.2 COMPROMISO DE BIT: PROBLEMA DEL RESIDUO CUADRÁTICO

Como herramienta para las Pruebas de Conocimiento Cero Computacionales, vimos los esquemas de compromiso basados en distintos problemas. Mostramos aquí una implementación del esquema de compromiso con vinculación incondicional basado en residuos cuadráticos 5.32.

Listing 3: Compromiso de bit con QR

```
y = 0

# Accion de P. Configuracion de parametros iniciales.

def Setup():
  p = random_prime(2^100)
  q = random_prime(2^100)
  N = p*q
  a = Zmod(p).random_element()
  while a.is_square():
    a = Zmod(p).random_element()
  b = Zmod(q).random_element()
  while b.is_square():
    b = Zmod(q).random_element()
  s = crt([ZZ(a),ZZ(b)],[p,q])
  #print kronecker(s,N)
  return s,N

# Hiding.

def Hiding(s,N,b):
  global y
  y = Zmod(N).random_element()
  while gcd(y,N)!=1:
    y = Zmod(N).random_element()
  x = Zmod(N)(s^b*y^2)
  return x

# Opening.

def v(s,N,x,y):
```

```
  if Zmod(N)(x-y^2)==0:
    return 0
  elif Zmod(N)(x-s*y^2)==0:
    return 1
  else:
    return -1

s,N = Setup()
x = Hiding(s,N,1)
print v(s,N,x,y)
```

En la etapa de `Setup` elegimos el módulo compuesto $N = pq$, y construimos el no-residuo cuadrático con símbolo de Jacobi 1, $s \in \mathbb{Z}_N^{Q^-}$, encontrando primero dos no-residuo cuadráticos módulo p y q respectivamente (símbolo de Legendre -1), y los combinamos utilizando el Teorema Chino de los Restos.

En la etapa de *ocultamiento*, elegimos el valor aleatorio $y \in \mathbb{Z}_N^*$, y calculamos el blob $x = s^b y^2 \bmod N$.

En la *apertura*, la función v recibe como parámetro el valor aleatorio y que abre el blob x.

B

IDENTITY MIXER PROTOCOL

En este anexo incluimos por completitud las pruebas de conocimiento cero no interactivas utilizadas durante la emisión de un certificado de Identity Mixer (Subsección 6.3.5).

Issuer Secret: p, q	Sys. pars. $(m_1, \ldots, m_l)$	**User** Secret: m_0
		Random v' $U := S^{v'} R_0^{m_0} \pmod n$
	$\xleftarrow{U, PK}$	$PK\{(\nu', \mu_0) : U \equiv S^{\nu'} R_0^{\mu_0} \pmod n\}$
Random v'' and prime e $A := \left(\frac{Z}{US^{v''}\prod_1^l R_i^{m_i}}\right)^{1/e} \pmod n$		
$PK\{(\delta) : A \equiv \left(\frac{Z}{US^{v''}\prod_1^l R_i^{m_i}}\right)^{\delta} \pmod n\}$	$\xrightarrow{(A,e,v''),PK}$	$v := v' + v''$ in signature (A, e, v)
		$Z \stackrel{?}{\equiv} A^e S^v \prod_0^l R_i^{m_i} \pmod n$

Figura 1: Idemix: firma de credenciales simplificada.

La prueba de conocimiento P_1 indica que conocemos un ν', primera parte del v final de la firma CL, y un m_0, nuestra clave secreta, tales que U se representa como (ν', m_0) en la *base* (S, R_0). Utilizando la heurística de Fiat-Shamir, en vez de pedir varios retos al Emisor para conseguir robustez, utilizamos una función de hash junto a un reto n_1 del emisor, llamado en inglés *nonce*. La prueba que se envía consiste en una tupla con el reto obtenido por el hash, y las respuestas a dicho reto.

Algoritmo B.1 ($KP\{(\nu', m_0) : U \equiv S^{\nu'} R_0^{m_0} \bmod n\}(n_1)$)**.**

1. Elegir $\tilde{m}_0$ y $\tilde{\nu}'$ aleatorios.
2. Calcular $\tilde{U} = S^{\tilde{\nu}'} \cdot R_0^{\tilde{m}_0} \bmod n$.
3. Reto por heurística Fiat-Shamir: $c = H(U \| \tilde{U} \| n_1)$.
4. Respuestas al *reto*: $\hat{\nu}' = \tilde{\nu}' + c\nu'$, $\hat{m}_0 = \tilde{m}_0 + cm_0$.
5. $P_1 := (c, \hat{\nu}', \hat{m}_0)$.

Para verificar la prueba anterior, por la heurística de Fiat-Shamir (6.1), debemos *recuperar* el valor de U calculado por el Usuario, y comparar los hashes generados.

Algoritmo B.2 (Verificar $P_1 := (c, \hat{\nu}', \hat{m}_0)(n_1)$)**.**

1. Calcular $\hat{U} = U^{-c} \cdot S^{\hat{\nu}'} R_0^{\hat{m}_0} \bmod n$.
2. Aceptar si $c = H(U \| \hat{U} \| n_1)$.

En la prueba de conocimiento P_2 del Emisor, demostramos que conocemos la inversa del e de la firma CL, pero sin desvelar su valor, que solo el Emisor, conocedor de p y q, puede calcular.

Algoritmo B.3 ($KP\{(e^{-1}) : A \equiv \left(\frac{Z}{U \cdot S^{v''} \cdot \prod_{i=1}^{l} R_i^{m_i}}\right)^{e^{-1}} \bmod n\}(n_2)$).

Llamemos $Q := \frac{Z}{U \cdot S^{v''} \cdot \prod_{i=1}^{l} R_i^{m_i}}$.

1. Elegir r aleatorio.
2. Calcular $\tilde{A} = Q^r \bmod n$.
3. Calcular el reto por Fiat-Shamir $c' = H(Q||A||n_2||\tilde{A})$.
4. Calcular la respuesta $s_e = r - c'e^{-1}$.
5. $P_2 := (c', s_e)$.

Como antes, para verificar la prueba, *recuperamos* el valor $\tilde{A}$ y comparamos los hashes.

Algoritmo B.4 (Verificar $P_2 := (c', s_e)(n_2)$).

1. Calcular $\hat{A} = A^{c'+s_e \cdot e} \equiv A^{c'} Q^{s_e} \bmod n$.
2. Aceptar si $c' = H(Q||A||n_2||\hat{A})$.

BIBLIOGRAFÍA

[1] László Babai. "Graph isomorphism in quasipolynomial time [extended abstract]". En: *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*. Ed. por Daniel Wichs y Yishay Mansour. ACM, 2016, págs. 684-697. ISBN: 978-1-4503-4132-5. DOI: 10.1145/2897518.2897542. URL: http://doi.acm.org/10.1145/2897518.2897542.

[2] Michael Ben-Or, Shafi Goldwasser y Avi Wigderson. "Completeness theorems for non-cryptographic fault-tolerant distributed computation". En: *Proceedings of the twentieth annual ACM symposium on Theory of computing*. ACM. 1988, págs. 1-10.

[3] Manuel Blum. "How to Prove a Theorem So No One Else Can Claim It". En: (1986).

[4] Jan Camenisch y Anna Lysyanskaya. "An efficient system for non-transferable anonymous credentials with optional anonymity revocation". En: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2001, págs. 93-118.

[5] Jan Camenisch y Anna Lysyanskaya. "A signature scheme with efficient protocols". En: *International Conference on Security in Communication Networks*. Springer. 2002, págs. 268-289.

[6] Jan Camenisch y Markus Stadler. "Efficient group signature schemes for large groups". En: *Advances in Cryptology—CRYPTO'97* (1997), págs. 410-424.

[7] David Chaum, Claude Crépeau e Ivan Damgard. "Multiparty unconditionally secure protocols". En: *Proceedings of the twentieth annual ACM symposium on Theory of computing*. ACM. 1988, págs. 11-19.

[8] Investigación y Ciencia, ed. *Resuelto el problema del isomorfismo de grafos (otra vez)*. http://www.investigacionyciencia.es/noticias/resuelto-el-problema-del-isomorfismo-de-grafos-otra-vez-14910.

[9] Claude Crépeau. "Efficient cryptographic protocols based on noisy channels". En: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 1997, págs. 306-317.

[10] Claude Crépeau y Joe Kilian. "Achieving oblivious transfer using weakened security assumptions". En: *Foundations of Computer Science, 1988., 29th Annual Symposium on*. IEEE. 1988, págs. 42-52.

[11] Ivan Damgard y Jesper Buus Nielsen. *Commitment Schemes and Zero-Knowledge Protocols (2011)*.

[12] Louis C. Guillou Jean-Jacques Quisquater y Thomas A. Berson. "How to Explain Zero-Knowledge Protocols to Your Children". En: *Advances in Cryptology - CRYPTO '89* (1990). Proceedings 435: 628-631. http://pages.cs.wisc.edu/~mkowalcz/628.pdf.

[13] Adrián Sánchez Martínez. *La cueva.* Imagen.

[14] Alfred J Menezes, Paul C Van Oorschot y Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[15] José Luis Gómez Pardo. *Introduction to Cryptography with Maple.* Springer Verlag, 2013. ISBN: 978-3-642-32165-8.

[16] Josef Pieprzyk, Thomas Hardjono y Jennifer Seberry. *Fundamentals of computer security*. Springer Science & Business Media, 2013.

[17] *Specification of the Identity Mixer Cryptographic Library (v2.3.43)*. Inf. téc. IBM Research, ene. de 2013.

[18] William Stallings. *Cryptography and Network Security: Principles and Practice*. 6.ª ed. Pearson Ed. Inc., 2014. ISBN: 978-0-13-335469-0.

[19] Douglas R Stinson. *Cryptography: theory and practice*. CRC press, 2005.

[20] Douglas R. Stinson. "Discrete mathematics and its applications". En: (2006). Ed. por Kenneth H Rosen.

[21] Ronald Rivest anb Clifford Stein Thomas Cormen Charles Leiserson. *Introduction to Algoritms.* Third Edition. The MIT Press, Cambridge, Massachusetts, 2009. ISBN: 978-0-262-03384-8.

ÍNDICE ALFABÉTICO