Trabajo de fin de grado

Universidad de Murcia

Facultad de Matemáticas

INTERCAMBIO PÚBLICO DE CLAVES USANDO MATRICES SOBRE ANILLOS DE GRUPO

Autor:
Eva Mª. Alarcón Díaz

Supervisor:
Dr. Ángel del Río Mateos

Septiembre 2015

Agradecimientos

Me gustaría que estas líneas sirvieran para expresar todo el agradecimiento que siento hacia las personas que me han acompañado a lo largo de la carrera y a las que han colaborado con la elaboración de este trabajo, porque gracias a vosotros estoy a punto de cumplir mi sueño: ser matemática.

En primer lugar, me gustaría dar las gracias a mi tutor Ángel del Río por su apoyo y por todo el tiempo y mensajes dedicados. También me gustaría agradecer al profesor Manuel Ruiz sus consejos en la parte de Estadística.

Por otro lado, gracias a mi familia por todo el apoyo que me han dado: a mi madre por enseñarme a esforzarme en conseguir lo que quiera y a no rendirme nunca, a mi padre por ser la primera persona que me enseñó a contar manzanas en los árboles ayudándome a entender mis primeros problemas de Matemáticas, y a Dani por ser un hermano que siempre está ahí y que me hace sentir orgullosa de ser su hermana.

No podría olvidar tampoco dar las gracias al Super Equipo 2 por todas esas tardes trabajando con Pokémon sin parar de reír y por hacerme sentir entre amigos de verdad, así que gracias, Sara y David. En particular, gracias, David por ser el mago que consigue explicarme Geometría con los Geomag, por tu paciencia infinita a pesar de mis frases de "no lo entiendo" y por ayudarme a no hundirme a pesar de las tormentas, tú has sido mi luz durante estos años.

A todos vosotros, muchas gracias.

Índice general

Abstract

Nowadays, Cryptography is constantly part of our lifes. For example, when we send an email or make an electronic transaction, we are transmiting private data which should not be accessible to unauthorized people. Therefore, whenever we perform these actions, *cryptosystems*, that is, algorithms that guarantee secrecy, are put into motion. However, technology and information technology progress at high speed, making obsolete a cryptosystem which was asumed to be secure when a new cryptoanalysis method is able to break it. This makes necessary a continuous research in order to find new methods to ensure the protection of information.

The aim of this Final Degree Project is the study of a new protocol proposed by Delaram Kahrobaei, Charalambos Koupparis and Vladimir Shpilrain in 2013 and published in the journal *Groups, Complexity and Cryptology* with the title *Public Key Exchange Using Matrices Over Group Rings* [13]. In this paper, the authors proposed a new public key exchange cryptographic protocol and made several experiments to test that their new method is useful. Roughly speaking, when two people try to send each other a message using a cryptosystem, they need to use a *key* to encrypt the information and both sides should know this key. To ensure the security of a cryptosystem it is customary to change the shared key from time to time, and it is at this moment that comes the paper of Kahrobaei, Koupparis y Shpilrain, since they propose a new method for agreeing on this key. In practice, often the parties that have to agree on a key are not close to each other and hence the information must be transmited through channels that are sometimes insecure. This led in 1976 to the first key exchange protocol in public channels, which is known as the *Diffie-Hellman Key Exchange Protocol* and was published in the paper *New Directions in Cryptography* [6]. Additionally, this paper helps to the appearance, later, of Public Key Cryptography. The paper that we are going to study is in the spirit of the Diffie-Hellman Protocol but using non-commutative rings instead of integers as it was the case in the Diffie-Hellman Protocol. More precisaly, the new protocol uses matrices over group rings. Therefore, throughout the project we will learn (or remember) some concepts that will be needed for, in the end, understanding the new proposed protocol and what the authors say on the paper about it.

The project is divided into four chapters and an appendix. The first chapter will be an introduction to Cryptography in general and it will provide us the general framework of the project. In this chapter, we will define some expresions that are commonly used in Cryptography and we will explain an example of a classic cryptosystem called Caesar's Cipher. We will also see one of the most famous ways to try to break this cryptosytem, the Frequency Analysis.

In the second chapter we will examine preliminary concepts that are necessary to understand the paper. This chapter will be divided into two sections, one for Algebra and one for Cryptography. In the first section we will recall basic background used throughout the Bachelor in Mathematics such as elementary properties of integers (specifically the Euclidean Algorithm, the Diophantine Equations and congruences) and the group and ring definitions. Also, we will define the group rings, wich are of great importance because the protocol proposed by the

authors uses them as we have mentioned. Finally, we will recall some definitions of Non Commutative Algebra that will be necessary to understand two important theorems of this branch of mathematics: the Wedderburn-Artin Theorem and the Maschke's Theorem. In the second section of the chapter we will continue introducing cryptographic concepts, including the Key Exchange Protocol of Diffie-Hellman. Moreover, we will put into mathematical words the expression "this algorithm take little time".

In the third chapter we will introduce the Discrete Logarithm Problem. This problem consist in, given a semigroup S, which is a non-empty set endowed with an associative operation, and two elements $a, b \in S$, find an integer x satisfying the equation $a^x = b$. At a first glance we can see that this problem can be very difficult to solve if, for example, the number of elements of S is very large. For that, there are several methods that try to solve this problem efficiently. In the chapter we will deal with four of the most important methods to solve the Discrete Logarithm Problem and also we will see the relation between that problem and another problem that is called the Diffie-Hellman Problem, since this last problem consists in breaking the Diffie-Hellman Key Exchange Protocol.

In the fourth chapter we will explain the contain of the paper of Kahrobaei, Koupparis and Shpilrain. First we will see why the authors believe that a new protocol is useful and what are the benefits that they observe in their new proposal. Then, we will analyze the effectiveness of the protocol taking into account various considerations. In first place, we will verify if the protocol finishes in small time, since it would not make sense to use it if it takes too long doing the calculations. After seeing that, indeed, takes little time, we will check if two assumptions are satisfied, they are called the Computational Diffie-Hellman and the Decision Diffie-Hellman assumptions. We will do this with three experiments. Having seen that, as the effectiveness of the protocol is based on the difficulty of solving the Discrete Logarithm Problem, we will want to check that, under the terms of the protocol and given a matrix M and a integer a relatively large, there is not a small integer c such that $M^a = M^c$. Otherwise we will say that M has a short cycle. However, if for example M is the identity matrix this c exists. So actually, we will check if, given M and a, the probability of the existence of c is very low. Finally, we will try to understand why the authors of the paper say that the most common attacks do not work on their protocol. At this moment I have to say that at one point, the authors refer to attacks with Quantum Algorithms and they say that those attacks are ineffective. However, due to the complexity of these algorithms, I could not understand these arguments and therefore I have not included it in this project. Finally and after having performed the experiments, we will emphasise that we can not ensure that the protocol proposed by the authors is one hundred percent effective because, as it is evident, we have not tested all possible methods and it could also happen that someone found a method able to break the protocol. In fact, during July of this year it has been announced in a paper a way to break the protocol proposed with methods of Non Commutative Algebra and Characters Theory (one of these methods is based in the Wedderbun-Artin Theorem that we will see in this project). The name of this paper is *Cryptanalysis of a System Using Matrices Over Group Rings*, but we won't deal with it because it could be the matter of another Final Degree Project.

The last part of this project consists on an appendix including the code of several programs that I have created to perform the experiments. In order to create this programs I have had to learn a completely new language for me, the language of the software GAP [9]. Therefore, I have thought that they are important because they are part of the work that I have had to do for this Final Degree Project.

I must finish mentioning that studying the paper of D. Kahrobaei, C. Koupparis and V. Shpilrain we have come up with three ideas that maybe can be useful for understanding better

the proposed protocol. The first idea refers to experiment 3 that we will see in 4.2.2: we believe that the results of the experiment proposed by the authors are not sufficient to support their conclusions. We base this opinion on a comparison of histograms of the analyzed data. The other two ideas serve to study the existence of short cycles by an alternative and more efficient way than the one used by the authors in the paper. In practice, in that paper they are not able to calculate the lengths of these cycles and simply perform some experiments that attempt to conclude that there are few short cycles. To obtain more accurate information on the lenght of the cycles we can use the Wedderburn Decomposition of the group ring which we are working. This is the second idea. The third idea is that we could optimize this analysis using the canonical form of the matrices appearing, although we have not had time yet to fully develop this last idea. We explain the second idea and the main lines of the third idea in 4.2.3.

Resumen

Hoy en día, la Criptografía forma parte de nuestras vidas en todo momento. Por ejemplo, cuando enviamos un correo electrónico o realizamos una transacción bancaria estamos enviando datos privados que no deseamos que obtenga una persona no autorizada. Por eso, cada vez que llevamos a cabo dichas acciones se ponen en marcha *criptosistemas*, es decir, conjuntos de procedimientos que garantizan la seguridad de la información. Sin embargo, la tecnología y la informática avanzan a gran velocidad, lo que hace que un criptosistema que había sido muy seguro de pronto deje de serlo y pase a ser muy sencillo de romperse, es decir, de obtener los datos que supuestamente protegía. Esto hace necesario investigar continuamente para tratar de encontrar nuevos métodos que garanticen la protección de la información y que sean infalibles contra métodos que intenten romperlos.

El objetivo de este Trabajo Fin de Grado es el estudio del fruto de una de esas investigaciones. En concreto, la investigación de Delaram Kahrobaei, Charalambos Koupparis y Vladimir Shpilrain que les llevó a publicar, en el año 2013, un artículo en la revista *Groups, Complexity, and Cryptology* titulado *Public Key Exchange Using Matrices Over Group Rings* [13]. En este artículo, los autores propusieron un nuevo protocolo criptográfico de intercambio de clave pública e hicieron varios experimentos para probar que su nuevo método era útil. A grandes rasgos, cuando dos personas tratan de enviarse un mensaje mediante un criptosistema, un elemento que ayuda enormemente a que solamente ellos sean capaces de obtener la información es la llamada *clave* del criptosistema. Para asegurar la seguridad de un criptosistema suele ser aconsejable cambiar la clave cada cierto tiempo, y es en este momento en el que entra en escena el artículo de D. Kahrobaei, C. Koupparis y V. Shpilrain, ya que propone un nuevo método para ponerse de acuerdo en dicha clave. En la práctica, a menudo los sujetos que tienen que ponerse de acuerdo en una clave no están próximos entre sí y para acordar dicha clave tienen que transmitirse la información por canales que a veces no son seguros. Esto dio lugar en 1976 al primer protocolo de intercambio de claves en canales públicos, que es conocido como el *Protocolo de Intercambio de Claves de Diffie-Hellman* y que fue publicado en el artículo *New Directions in Cryptography* [6]. Además, este artículo sentó las bases para el nacimiento, más adelante, de la Criptografía de Clave Pública. El artículo que vamos a estudiar es una modificación del Protocolo de Diffie-Hellman en el que se utilizan anillos no conmutativos en vez de usar los números enteros como en dicho protocolo. Más concretamente, usaremos matrices sobre anillos de grupo. Por tanto, a lo largo del trabajo necesitaremos ir aprendiendo (o recordando) conceptos que serán necesarios para, al final, comprender el nuevo protocolo planteado y lo que dicen sus autores sobre él en el artículo.

El trabajo se divide en cuatro capítulos y un anexo. El primer capítulo servirá como introducción a la Criptografía en general y nos pondrá en situación con respecto al artículo: en él definiremos algunas expresiones comúnmente usadas en Criptografía y explicaremos un ejemplo de criptosistema clásico llamado Criptosistema de César. También veremos una de las formas más famosas para intentar romper dicho criptosistema, el Análisis de Frecuencias.

Con el segundo capítulo pasaremos a estudiar conceptos preliminares necesarios para entender el artículo. Este capítulo estará dividido en dos secciones, una de Álgebra y una de Criptografía. En la primera sección recordaremos conocimientos básicos usados a lo largo del Grado en Matemáticas tales como algunas propiedades elementales de los números enteros (como son el Algoritmo de Euclides, las Ecuaciones Diofánticas y las congruencias) o la definición de grupo y de anillo. Además, definiremos los anillos de grupo, los cuales tienen una gran importancia porque el protocolo propuesto por los autores los usa de forma directa como ya hemos comentado. Por último, recordaremos algunas definiciones de la asignatura de Álgebra No Conmutativa que serán necesarias para entender dos grandes teoremas de esta rama de las matemáticas: el Teorema de Wedderburn-Artin y el Teorema de Maschke. En la segunda sección del capítulo continuaremos introduciendo conceptos de Criptografía y, entre ellos, estudiaremos el Protocolo de Intercambio de Claves de Diffie-Hellman y expresaremos de forma matemática el concepto de que un algoritmo "tarde poco".

En el tercer capítulo nos adentraremos en el Problema del Logaritmo Discreto. Este problema consiste en, dado un semigrupo S (que no es más que un conjunto no vacío dotado con una operación asociativa) y dos elementos $a, b \in S$, encontrar un entero x que satisfaga la ecuación $a^x = b$. A simple vista podemos observar que este problema puede ser muy complicado de resolver si por ejemplo el número de elementos de S es muy grande. Debido a ello, existen diversos métodos para intentar resolver este problema y que tratan de ser lo más eficientes posible. En el capítulo abordaremos cuatro métodos de los más importantes para resolver el Problema del Logaritmo Discreto y también veremos la relación existente entre este problema y el llamado Problema de Diffie-Hellman, ya que éste último problema consiste en romper el Protocolo de Intercambio de Claves de Diffie-Hellman.

Una vez comprendidos los capítulos anteriores, en el cuarto capítulo pasaremos a explicar los contenidos del artículo de Kahrobaei, Koupparis y Shpilrain. Primero veremos por qué los autores se plantean la necesidad de un nuevo protocolo y cuáles son las ventajas que le ven al suyo. Después, analizaremos la eficacia del protocolo teniendo en cuenta distintas consideraciones. En primer lugar veremos que el tiempo que se tarda en llevar a cabo el protocolo es relativamente pequeño, puesto que no tendría mucho sentido usarlo si tardase demasiado tiempo en hacer los cálculos. Después de ver que, efectivamente, tarda poco tiempo, pasaremos a comprobar si se satisfacen dos hipótesis llamadas Hipótesis Computacional e Hipótesis de Decisión de Diffie-Hellman, esto lo haremos mediante tres experimentos. Una vez visto eso, como la eficacia del protocolo está basada en la dificultad de resolver el Problema del Logaritmo Discreto, comprobaremos si se cumple que, bajo las condiciones del protocolo y dada una matriz M y un entero a relativamente grande, no existe un entero c mucho menor que a tal que $M^a = M^c$. Si existiera dicho c diríamos que M tiene un ciclo corto. Sin embargo, si por ejemplo M es la matriz identidad podemos observar que claramente existe ese c. Entonces, lo que en realidad acabaremos comprobando es si la probabilidad de encontrar c dados M y a es muy baja. Por último, veremos por qué dicen los autores que los principales ataques a protocolos no sirven contra el suyo. En este punto he de decir que en cierto momento los autores del artículo hacen referencia a que han estudiado posibles ataques por medio de Algoritmos Cuánticos y que son ineficaces contra su protocolo. Sin embargo, debido a la complejidad de dichos algoritmos no he podido comprender esos argumentos y por tanto no los he incluido en este trabajo. Finalmente, y tras haber realizado los experimentos, recalcaremos que no podemos asegurar que el protocolo propuesto por los autores sea cien por cien eficaz, ya que, como es evidente, no hemos comprobado todos los métodos posibles y además podría suceder que alguien encontrara un método nuevo que consiga romper el protocolo. De hecho, en julio de este mismo año se ha publicado un artículo llamado *Cryptanalysis of a System Using Matrices Over Group Rings* en el que los autores dicen haber encontrado una manera de romper el código de forma sencilla, entendiendo

por “forma sencilla” usar conceptos de Álgebra No Conmutativa y de Teoría de Caracteres (uno de esos conceptos es el Teorema de Wedderburn-Artin y que sí veremos en este trabajo), pero este artículo sería por sí solo un tema para otro posible trabajo, por lo que simplemente lo mencionaré sin dar detalles.

La última parte de este trabajo consiste en un anexo que incluye los códigos de varios programas que he tenido que ir creando para poder llevar a cabo los experimentos. Para crearlos he necesitado aprender desde cero el lenguaje del software GAP [9]. Por tanto, he creído necesario incluir dichos códigos porque considero que también forman parte del trabajo realizado para este Trabajo Fin de Grado.

Por último debo decir que estudiando el artículo de Kahrobaei, Koupparis y Shpilrain se nos han ocurrido tres ideas que pueden servir para la mejor comprensión del protocolo propuesto. La primera de ellas se refiere al experimento 3 que vemos en 4.2.2: creemos que los resultados del experimento que proponen los autores no son suficientes para soportar las conclusiones a las que llegan ellos. Basamos esta opinión en una comparación de histogramas de los datos analizados. Las otras dos ideas sirven para estudiar la existencia de ciclos cortos de forma alternativa y más eficiente a como lo hacen en el artículo que estamos estudiando. En la práctica, en dicho artículo no son capaces de calcular las longitudes de esos ciclos y simplemente realizan algunos experimentos de los que intentan concluir que hay pocos ciclos cortos. Para obtener información más precisa sobre la longitud de los ciclos podemos utilizar la Descomposición de Wedderburn del anillo de grupo con el que estamos trabajando. Ésta es la segunda idea. Con la tercera idea podríamos optimizar este análisis utilizando la forma canónica de las matrices que aparecen, aunque todavía no hemos tenido tiempo de desarrollar completamente esta última idea. Tanto la segunda idea como las líneas principales de la tercera idea están explicadas en 4.2.3.

Capítulo 1

Introducción

A lo largo de la historia, el ser humano ha tenido la necesidad de comunicarse. Al principio, cuando el mensaje era enviado mediante un mensajero y era de poca importancia no importaba mucho si era interceptado o no y un tercero lo leía, pero poco a poco se hizo necesario ocultar de alguna manera la información de los mensajes enviados por si el enemigo interceptaba el mensaje. Fue así como nació la *Criptografía*, que toma su nombre de los términos griegos *"krypto"* y *"graphos"* que significan, respectivamente, oculto y escritura. Por tanto, el objetivo principal de la Criptografía es mantener la privacidad de la comunicación entre dos o más personas transformando la información del mensaje de forma que si una tercera persona lee el mensaje transformado no sea capaz de conocer el mensaje original. A menudo, a la acción de transformar el mensaje original (llamado *texto en claro*) se le llama *encriptar* o *cifrar*, mientras que a la acción inversa, es decir, dado el mensaje transformado obtener el mensaje original, se le llama *desencriptar* o *descifrar*.

Poco a poco, de forma paralela al avance de la Criptografía fueron surgiendo el *Criptoanálisis* y los *criptoanalistas*, es decir, personas que estudian métodos para obtener el texto en claro sin tener acceso a la información secreta necesaria para descifrar el mensaje (llamada *clave*). En el caso de conseguirlo, diremos que el criptoanalista ha *roto el código*. Llamaremos *Criptología* al estudio de la Criptografía y el Criptoanálisis.

Dado un mensaje cifrado existen dos tipos básicos de amenazas, la *amenaza pasiva* y la *amenaza activa*. En el primer caso, el enemigo sólamente quiere averiguar el contenido del mensaje, mientras que en el segundo caso quiere modificar o falsear el contenido del mensaje. En este trabajo siempre supondremos que el enemigo es una amenaza pasiva. Por otro lado, la seguridad de los métodos usados para cifrar mensajes puede dividirse en varios tipos:

- *Seguridad teórica o perfecta*: el método es seguro contra cualquier enemigo que tenga recursos computacionales y tiempo ilimitados.
- *Seguridad práctica o computacional*: el método es seguro contra los enemigos que tengan insuficiente tiempo y/o recursos computacionales.
- *Seguridad probable*: aún no se ha demostrado que el método sea seguro pero hasta ahora nadie ha conseguido romper el código.
- *Seguridad condicional*: el método es seguro hasta que se desarrollen nuevos (y mejores) métodos de criptoanálisis.

Vamos a ver un ejemplo muy sencillo de cómo cifrar un mensaje: el llamado *Criptosistema de César*. Para cifrar un mensaje mediante este método, se escribe el alfabeto en el orden usual y debajo de él se escribe de nuevo el alfabeto pero desplazando las letras por ejemplo 3

espacios a la derecha, de forma que la letra A se transforma en la letra D, la letra B en la E y así sucesivamente. Vamos a ver cómo quedaría:

Alfabeto usual:	A B C D E F G H I J K L M N Ñ O P Q R S T U V W X Y Z
Alfabeto cifrado:	X Y Z A B C D E F G H I J K L M N Ñ O P Q R S T U V W

De esta manera, si quisiéramos enviar el mensaje "hoy va a llover", lo cifraríamos de la siguiente forma omitiendo tildes y espacios:

$$\text{HOYVAALLOVER} \longrightarrow \text{EMVSXXIIMSBO}$$

Existen múltiples formas de descifrar estos mensajes a pesar de no conocer cuántas veces se ha desplazado el alfabeto. Una de ellas es el *Análisis de Frecuencias*, es decir, se cogen todas las letras del alfabeto español y se ordenan de mayor a menor frecuencia de aparición (E, A, O, etc.). Después, se mira el mensaje cifrado y a la letra que aparezca más veces en él se le asigna la primera letra más frecuente del alfabeto, a la segunda que aparezca más veces se le asigna la segunda letra más frecuente, y así sucesivamente. Lógicamente, este método será más efectivo cuando el mensaje sea muy largo, sin embargo, con esta forma de encriptar mensajes no es muy difícil obtener la clave simplemente probando con todas las opciones posibles.

Tristemente, la Criptología ha ido avanzando a lo largo de la historia sobre todo debido a las constantes guerras que ha llevado a cabo la humanidad desde el principio de los tiempos, ya que en éstas es de suma importancia comunicarse con los aliados y enviar órdenes sin que el enemigo pueda obtener dicha información y también tratar de obtener información de los mensajes del enemigo. Los ejemplos más conocidos son la famosa máquina Enigma usada por los nazis para cifrar mensajes durante la Segunda Guerra Mundial y la máquina Colossus que fue usada por los británicos para descifrar esos mensajes.

El mundo va evolucionando tecnológicamente y a gran velocidad, por lo que es necesario buscar continuamente nuevas formas de cifrar mensajes que un enemigo con un potente ordenador sea incapaz de descifrar. De esta manera, en 1976, Whitfield Diffie (nacido en 1944) y Martin Edward Hellman (nacido en 1945) publicaron un artículo llamado *New Directions in Cryptography* [6] en el cual proponían un nuevo método con el que poder intercambiar claves sin que una tercera persona externa pudiera conseguir dicha información. Acababa de empezar la Criptografía de Clave Pública. Sin embargo, desde 1976 los ordenadores han avanzado enormemente y han surgido nuevos métodos, en concreto, el método de intercambio de claves explicado en el artículo *Public Key Exchange Using Matrices Over Group Rings* [13], publicado en 2013 por Delaram Kahrobaei, Charalambos Koupparis y Vladimir Shpilrain y motivo de este trabajo.

Capítulo 2

Preliminares

Antes de poder sumergirnos en el nuevo método de D. Kahrobaei, C. Koupparis y V. Shpilrain, vamos a explicar algunos conceptos de Álgebra y de Criptografía. Las referencias usadas en este capítulo son [4], [13], [18] y [21] para la Sección 2.1, y [14], [16] y [20] para la Sección 2.2.

En este capítulo no incluiremos demostraciones porque solo nos interesa recopilar los conceptos y resultados necesarios para los capítulos posteriores. Para los resultados que no sean bien conocidos incluiremos referencias de dónde aparecen las demostraciones o una idea de dichas demostraciones sin entrar en muchos detalles.

2.1. Preliminares de Álgebra

Vamos a comenzar viendo algunos conceptos de Álgebra que serán útiles a lo largo de este trabajo.

2.1.1. Aritmética elemental

En esta sección vamos a repasar algunos conceptos básicos de los números enteros que nos serán muy útiles sobre todo en el Capítulo 3.

Algoritmo de Euclides

Si recordamos, la definición de *máximo común divisor* de dos números enteros a y b es la siguiente: el mayor entero d tal que $d \mid a$ y $d \mid b$. Pero, ¿cómo calculamos ese número d? Euclides ya conocía una forma de hacerlo que plasmó en su obra *Los Elementos*. El *Algoritmo de Euclides* tal y como se conoce hoy en día es el que vamos a ver a continuación, aunque para ello vamos a ver la proposición en la que se basa dicho algoritmo.

Proposición 2.1.1. *Sean $a, b \in \mathbb{Z}$. Entonces, para todo $\alpha \in \mathbb{Z}$ se tiene*

$$\mathrm{mcd(a, b)} = \mathrm{mcd(a - \alpha b, b)} = \mathrm{mcd(a, b - \alpha a)}.$$

En particular, si $b \neq 0$ y $a = bq + r$ es la división entera de a entre b, tenemos que $\mathrm{mcd(a, b)} = \mathrm{mcd(b, r)}$.

El cálculo del máximo común divisor de dos enteros mediante el *Algoritmo de Euclides* se obtiene aplicando varias veces la Proposición 2.1.1. Podemos suponer que a y b son positivos y

entonces tenemos:

$$\begin{array}{lll} a = bq_1 + r_1 & \mathrm{mcd(a,b)} = \mathrm{mcd(b,r_1)} & \mathrm{r_1 < b} \\ b = r_1q_2 + r_2 & \mathrm{mcd(b,r_1)} = \mathrm{mcd(r_1,r_2)} & \mathrm{r_2 < r_1} \\ r_1 = r_2q_3 + r_3 & \mathrm{mcd(r_1,r_2)} = \mathrm{mcd(r_2,r_3)} & \mathrm{r_3 < r_2} \\ & \dots & \end{array} \tag{2.1}$$

Como $b > r_1 > r_2 > r_3 > \ldots \geq 0$ y todos son números enteros, debe obtenerse cero en un número finito de pasos:

$$\begin{array}{ll} r_{n-2} = r_{n-1}q_n + r_n & \mathrm{mcd(r_{n-2},r_{n-1})} = \mathrm{mcd(r_{n-1},r_n)} \\ r_{n-1} = r_nq_{n+1} & \mathrm{mcd(r_{n-1},r_n)} = \mathrm{r_n}. \end{array} \tag{2.2}$$

Luego $\mathrm{mcd(a,b)} = \mathrm{r_n}$.

Estos cálculos (2.1) y (2.2) se pueden disponer en forma de tabla de la manera siguiente:

	q_1	q_2	$\cdots$	$\cdots$	q_{n-1}	q_n	q_{n+1}
a	b	r_1	$\cdots$	$\cdots$	r_{n-2}	r_{n-1}	r_n
r_1	r_2	r_3	$\cdots$	$\cdots$	r_n	0	

Tabla 2.1: Algoritmo de Euclides.

Además, el método anterior también nos sirve para calcular dos enteros s y t tales que $d = sa + tb$, donde $d = r_n = \mathrm{mcd(a,b)}$. Esta igualdad se conoce como una *Identidad de Bézout*, llamada así por Étienne Bézout (1730-1783). El método consiste en escribir $d = r_n$ en función de a y b usando las ecuaciones $r_i = r_{i-2} - r_{i-1}q_i$ dadas por (2.1) y (2.2), donde i va tomando los valores $n, n-1, \ldots, 1$ y entendemos que $a = r_{-1}$ y $b = r_0$.

Ecuaciones Diofánticas

A continuación vamos a ver una importante aplicación del Algoritmo de Euclides. Se trata de la resolución de las llamadas *Ecuaciones Diofánticas*, es decir, las ecuaciones del tipo

$$aX + bY = c \tag{2.3}$$

donde a, b y c son números enteros con $a, b \neq 0$ y X e Y son incógnitas. El adjetivo de "diofántica" (en honor a Diofanto de Alejandría) significa que sólo nos interesan las soluciones de dicha ecuación que sean números enteros. Por tanto, una *solución* de la ecuación diofántica (2.3) es un par ordenado de números (x, y) tales que $ax + by = c$. Antes de ver qué forma tienen las soluciones necesitamos saber si existe alguna solución, pero esto nos lo da la siguiente proposición.

Proposición 2.1.2. *Supongamos que queremos resolver la ecuación diofántica (2.3) y que tenemos* $d = \mathrm{mcd(a,b)}$, $a' = a/d$ *y* $b' = b/d$. *Entonces, la ecuación tiene soluciones enteras si y sólo si* $d \,|\, c$. *En ese caso, si* $c' = c/d$, *las soluciones de (2.3) son las mismas que las de*

$$a'X + b'Y = c'.$$

Una vez vista esta proposición, para resolver una ecuación diofántica procederemos de la siguiente forma (obsérvese que en los casos en los que $\mathrm{mcd(a,b)} = 1$ o $c = 0$ será mucho más sencillo resolver la ecuación):

1. Calculamos $d = \mathrm{mcd(a, b)}$, por ejemplo usando el Algoritmo de Euclides.

2. Si d no divide a c, deducimos que la ecuación no tiene solución y acabamos.

3. Si d divide a c, calculamos una solución particular. Para ello, sea $c' = c/d$ y sean r y s dos enteros tales que $ar + bs = d$ (esto se puede hacer de nuevo con el Algoritmo de Euclides); entonces $(x_0, y_0) = (rc', sc')$ es una solución.

4. Si $a' = a/d$ y $b' = b/d$, entonces las soluciones de (2.3) son todos los pares

$$(x, y) = (x_0 + \lambda b', y_0 - \lambda a'),$$

donde λ es un entero arbitrario.

Congruencias

Dados tres números enteros a, b y n, diremos que a y b son *congruentes módulo* n y escribiremos $a \equiv b \mod n$ si $a - b$ es múltiplo de n.

Fijado n, la relación "ser congruente módulo n" es una relación de equivalencia (es decir, es reflexiva, simétrica y transitiva) que induce una partición de $\mathbb{Z}$ en clases de equivalencia. Denotaremos con $[a]_n$ la clase de a módulo n. Sin embargo, cuando no exista confusión, la denotaremos por $\overline{a}$. Por otro lado, como es habitual denotaremos por $\mathbb{Z}_n$ al conjunto de las clases de equivalencia.

Otra aplicación del Algoritmo de Euclides es la resolución de las *Ecuaciones Lineales de Congruencias*. Una *ecuación lineal de congruencias* no es más que una ecuación de la forma

$$aX \equiv c \mod n, \tag{2.4}$$

donde $n \neq 0$. Como en las ecuaciones diofánticas, sólo nos interesarán las soluciones de (2.4) que sean números enteros. La siguiente proposición nos dirá cuándo tienen solución estas ecuaciones.

Proposición 2.1.3. *Sean a y n enteros con $n \neq 0$, y sean $d = \mathrm{mcd(a, n)}$, $a' = a/d$ y $n' = n/d$. Entonces la ecuación (2.4) tiene solución si y solo si $d \mid c$. En este caso, si $c' = c/d$, las soluciones de (2.4) son las mismas que las de*

$$a'X \equiv c' \mod n'.$$

Si observamos, x será solución de la ecuación (2.4) si y sólo si el par (x, y) es solución para algún entero y de la ecuación (2.3) para $b = n$. Por tanto, sabemos calcular las soluciones de las ecuaciones lineales de congruencias remitiéndonos al caso de las ecuaciones diofánticas.

Una vez visto esto podemos adentrarnos en el Teorema Chino de los Restos. Este teorema nos será útil más adelante en el Capítulo 3.

Teorema 2.1.4 (Teorema Chino de los Restos)**.** *Sean $b_1, \ldots, b_k$ enteros arbitrarios y $m_1, \ldots, m_k$ enteros positivos tales que $\mathrm{mcd(m_i, m_j)} = 1$ para todo $i \neq j$. Entonces, el sistema de congruencias*

$$\begin{cases} x \equiv b_1 \mod m_1 \\ \quad \ldots \\ x \equiv b_k \mod m_k \end{cases} \tag{2.5}$$

tiene solución única módulo $M = m_1 \cdots m_k$.

Este teorema nos dice que el sistema de congruencias (2.5) tiene solución, pero además de saber que existe solución necesitaremos saber cómo calcularla. Vamos a verlo:

1. Calculamos $M = m_1 \cdot \ldots \cdot m_k$ y $M_i = M/m_i$.

2. Ahora, usando la Proposición 2.1.3, sabemos que cada M_i tiene un inverso módulo m_i. Sean N_i tales que $M_i N_i \equiv 1 \mod m_i$ con $i = 1, \ldots, k$.

3. El número entero $x_0 = b_1 M_1 N_1 + \ldots + b_k M_k N_k$ es solución del sistema (2.5) y por tanto las demás soluciones son de la forma $x_0 + \lambda M$ con λ un número entero.

2.1.2. Grupos y Anillos

Los conceptos que vamos a ver en esta segunda parte vuelven a ser conocidos, pero no por ello dejan de ser importantes.

Una *operación* o *ley de composición interna* en un conjunto A es una aplicación

$$* : A \times A \longrightarrow A.$$

La definición de operación es demasiado general, y en la práctica nos interesarán sólo las operaciones que verifiquen ciertas propiedades que las hagan útiles y manejables. Veamos algunas de las propiedades más importantes. Sea $*$ una operación en un conjunto no vacío A. Diremos que:

- La operación es *asociativa* si $a * (b * c) = (a * b) * c$ para cualesquiera $a, b, c \in A$.
- La operación es *conmutativa* si $a * b = b * a$ para cualesquiera $a, b \in A$.
- El elemento $e \in A$ es un *elemento neutro* para $*$ si $a * e = a = e * a$ para todo $a \in A$.
- Si existe un elemento neutro e, el elemento $b \in A$ es *simétrico* de $a \in A$ si $a * b = e = b * a$.

Con estos conceptos podemos dar la conocidas definiciones de grupo y de anillo. Sin embargo, antes veremos una estructura más sencilla, la estructura de semigrupo.

Definición 2.1.5. Un *semigrupo* es un par $(A, *)$ donde A es un conjunto no vacío y $*$ es una operación en A que es asociativa. Si la operación es conmutativa, diremos que es un *semigrupo conmutativo* o *abeliano*.

Definición 2.1.6. Un *grupo* es un par $(A, *)$ formado por un conjunto no vacío A y una operación $*$ en A que es asociativa, tiene elemento neutro e y cumple que cada elemento a de A tiene un simétrico. Si además la operación es conmutativa diremos que el grupo es *abeliano*.

Ejemplo 2.1.7. El conjunto de las unidades de $\mathbb{Z}_n$, es decir, $\mathbb{Z}_n^*$, es un grupo abeliano.

El siguiente ejemplo lo necesitaremos más adelante en 2.1.3.

Ejemplo 2.1.8. Dado un conjunto no vacío A, denotamos por $\mathcal{S}(A)$ al *grupo simétrico de A*. Es decir, $\mathcal{S}(A)$ está formado por todas las permutaciones de A junto con la operación "composición". La composición siempre es asociativa, el elemento neutro es la aplicación identidad y el inverso de una aplicación es su aplicación inversa en el sentido usual. Si tomamos $A = \mathbb{N}_n = \{1, \ldots, n\}$, esto se suele denotar por $\mathcal{S}_n$ y se le llama *grupo simétrico en n elementos*.

Notación 2.1.9. Por lo general existen dos notaciones para los grupos, la *notación aditiva* y la *notación multiplicativa*. En la *notación aditiva* llamaremos *cero* al elemento neutro y *opuesto* al elemento simétrico; por otro lado, en la *notación multiplicativa* llamaremos *uno* al neutro e *inverso* al simétrico. Las notaciones se pueden ver en la siguiente Tabla 2.2.

	Aditiva	Multiplicativa
$*$	$+$	$\cdot$
Neutro	0	1
Simétrico de a	$-a$	a^{-1}

Tabla 2.2: Notación aditiva y multiplicativa.

Lo más frecuente es usar la notación multiplicativa (sea el grupo abeliano o no), pero en el caso de que el grupo sea abeliano también se puede usar la notación aditiva.

Definición 2.1.10. Sea $(G, \cdot)$ un grupo. Un *subgrupo* de G es un subconjunto H de G cerrado para la operación $\cdot$ y tal que $(H, \cdot)$ es un grupo.

Dados dos grupos, existen unas aplicaciones que los relacionan llamadas *homomorfismos de grupos*.

Definición 2.1.11. Un *homomorfismo del grupo* $(G, \cdot)$ *en el grupo* $(H, *)$ es una aplicación $f : G \longrightarrow H$ que conserva la operación, es decir, que verifica $f(a \cdot b) = f(a) * f(b)$ para cualesquiera $a, b \in G$.

Definición 2.1.12. Un *anillo (con identidad)* es una terna $(A, +, \cdot)$ formada por un conjunto no vacío A y dos operaciones $+$ y $\cdot$ en A; la primera llamada usualmente *suma* y la segunda *producto* o *multiplicación*, que verifican:

1. A es un grupo abeliano con la suma.
2. El producto es asociativo y tiene un elemento neutro.
3. Se verifica la propiedad distributiva, es decir, dados $a, b, c \in A$ se cumple que

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c).$$

Se pueden definir *anillos conmutativos* simplemente añadiendo la condición de que el producto sea conmutativo.

En general, no se asume que dado un elemento de A tenga un elemento simétrico para el producto. Sin embargo, habrá casos en los que necesitaremos usar elementos que cumplan dicha propiedad, lo que da lugar a las definiciones de *elemento invertible* y de *cuerpo*.

Definición 2.1.13. Diremos que $a \in A$ es un elemento *invertible* o *unidad* en A si existe un elemento simétrico suyo para el producto. En ese caso, al elemento simétrico lo denotaremos por a^{-1} y lo llamaremos el *inverso* de a. Denotaremos por A^* al conjunto de todas las unidades de A. Se dice que un anillo conmutativo A es un *cuerpo* si tiene todos sus elementos no nulos invertibles.

Ejemplos 2.1.14.

- El conjunto $\mathbb{Z}_n$ es un anillo conmutativo. Además, si n es primo (es decir, si los únicos enteros que dividen a n son el 1 y n), entonces como consecuencia de la Proposición 2.1.3 se tiene que el conjunto $\mathbb{Z}_n^*$ está formado por $n - 1$ elementos.
- El conjunto $\mathbb{R}$ de los números reales es un cuerpo.

- Si A es un anillo y n es un entero positivo, entonces el conjunto $\mathcal{M}_n(A)$ de las matrices de tamaño $n \times n$ con entradas en A forma un anillo con la suma y producto de matrices habituales.

La siguiente definición será necesaria en la parte de Álgebra No Conmutativa 2.1.4.

Definición 2.1.15. Sea A un anillo, y recordemos que si $n \in \mathbb{Z}^+$ escribimos $n \cdot 1 = 1 + \ldots + 1$ (n veces). Si existe $n \in \mathbb{Z}^+$ tal que $n \cdot 1 = 0$, definimos la *característica de* A como el menor $n \in \mathbb{Z}^+$ que verifica tal igualdad. Si no existe un tal n, decimos que la característica de A es 0. Denotaremos la característica de A por car(A).

Al igual que pasaba con los grupos, dados dos anillos existen un tipo de aplicaciones que los relacionan.

Definición 2.1.16. Sean A y B dos anillos. Un *homomorfismo de anillos* entre A y B es una aplicación $f : A \longrightarrow B$ que conserva las operaciones y la unidad; es decir, que satisface $f(x+y) = f(x)+f(y)$, $f(x \cdot y) = f(x) \cdot f(y)$ para cada par de elementos $x, y \in A$, y que cumple $f(1_A) = 1_B$.

2.1.3. Anillos de Grupos

Los conocimientos de esta parte serán esenciales para comprender el artículo *Public Key Exchange Using Matrices Over Group Rings* [13], pues el protocolo está basado en matrices sobre anillos de grupo.

Definición 2.1.17. Sea G un grupo multiplicativo y sea R un anillo con la unidad distinta del cero. Se define el *anillo de grupo* RG como el conjunto cuyos elementos son las sumas formales de la forma

$$\sum_{g \in G} r_g g$$

donde $r_g \in R$ y todos los r_g son cero salvo una cantidad finita de ellos.

A continuación, vamos a ver cómo se definen la suma y el producto en un anillo de grupo.

- Dados dos elementos de RG, definimos su suma como

$$\left(\sum_{g \in G} a_g g\right) + \left(\sum_{g \in G} b_g g\right) = \sum_{g \in G} (a_g + b_g)\, g$$

  Esta definición tiene sentido, ya que $a_g + b_g = 0$ para todos los $g \in G$ salvo para una cantidad finita, así que la suma está en RG. Además, esta suma es asociativa, simétrica y conmutativa, y para cada elemento de RG existe un elemento simétrico. Por tanto, $(RG, +)$ es un grupo abeliano.

- Dados dos elementos de RG, definimos su producto como

$$\left(\sum_{g \in G} a_g g\right)\left(\sum_{g \in G} b_g g\right) = \sum_{g \in G} \left(\sum_{f,h \in G, f \cdot h = g} a_f b_h\right) g$$

Veamos ahora un ejemplo de anillo de grupo y hagamos algunas operaciones con elementos suyos.

Ejemplo 2.1.18. Sea el anillo de grupo $\mathbb{Z}_7\mathcal{S}_5$ formado por el grupo el grupo simétrico $\mathcal{S}_5$ y el anillo $\mathbb{Z}_7$, y denotemos por e a la unidad de $\mathcal{S}_5$. Vamos a tomar dos elementos para ver cómo se sumarían y multiplicarían:

$$a = 6(1\ 3\ 5) + 3(2\ 5\ 3\ 4) + 5(1\ 4)(2\ 5\ 3)$$
$$b = 2(2\ 5) + 4(1\ 3\ 5) + (4\ 5)$$

Suma:

$$\begin{aligned} a+b &= [6(1\ 3\ 5) + 3(2\ 5\ 3\ 4) + 5(1\ 4)(2\ 5\ 3)] + [2(2\ 5) + 4(1\ 3\ 5) + (4\ 5)] \\ &= 10(1\ 3\ 5) + 3(2\ 5\ 3\ 4) + 5(1\ 4)(2\ 5\ 3) + 2(2\ 5) + (4\ 5) \\ &= 3(1\ 3\ 5) + 3(2\ 5\ 3\ 4) + 5(1\ 4)(2\ 5\ 3) + 2(2\ 5) + (4\ 5) \end{aligned}$$

Producto:

$$\begin{aligned} ab &= [6(1\ 3\ 5) + 3(2\ 5\ 3\ 4) + 5(1\ 4)(2\ 5\ 3)]\,[2(2\ 5) + 4(1\ 3\ 5) + (4\ 5)] \\ &= 12(1\ 3\ 5\ 2) + 24(1\ 5\ 3) + 6(1\ 3\ 5\ 4) + 6(2\ 3\ 4) + 12(1\ 4\ 2\ 5) \\ &\quad + 3(2\ 5)(3\ 4) + 10(1\ 4)(2\ 3) + 20(1\ 2\ 5\ 4) + 5(1\ 4\ 3\ 2\ 5) \\ &= 5(1\ 3\ 5\ 2) + 3(1\ 5\ 3) + 6(1\ 3\ 5\ 4) + 6(2\ 3\ 4) + 5(1\ 4\ 2\ 5) \\ &\quad + 3(2\ 5)(3\ 4) + 3(1\ 4)(2\ 3) + 6(1\ 2\ 5\ 4) + 5(1\ 4\ 3\ 2\ 5) \end{aligned} \tag{2.6}$$

$$\begin{aligned} ba &= [2(2\ 5) + 4(1\ 3\ 5) + (4\ 5)]\,[6(1\ 3\ 5) + 3(2\ 5\ 3\ 4) + 5(1\ 4)(2\ 5\ 3)] \\ &= 12(1\ 3\ 2\ 5) + 6(3\ 4\ 5) + 10(1\ 4)(3\ 5) + 24(1\ 5\ 3) + 12(1\ 3\ 4\ 2) \\ &\quad + 20(1\ 4\ 3\ 2) + 6(1\ 3\ 4\ 5) + 3(2\ 4)(3\ 5) + 5(1\ 5\ 3\ 2\ 4) \\ &= 5(1\ 3\ 2\ 5) + 6(3\ 4\ 5) + 3(1\ 4)(3\ 5) + 3(1\ 5\ 3) + 5(1\ 3\ 4\ 2) \\ &\quad + 6(1\ 4\ 3\ 2) + 6(1\ 3\ 4\ 5) + 3(2\ 4)(3\ 5) + 5(1\ 5\ 3\ 2\ 4) \end{aligned} \tag{2.7}$$

Una vez visto cómo funciona nuestro anillo de grupo, la definición de $\mathcal{M}_n(\mathbb{Z}_7\mathcal{S}_5)$, es decir, del anillo de las matrices de tamaño $n \times n$ sobre el anillo de grupo $\mathbb{Z}_7\mathcal{S}_5$, es clara. Aún así, veamos un pequeño ejemplo de cómo multiplicar elementos de $\mathcal{M}_2(\mathbb{Z}_7\mathcal{S}_5)$, que es la operación que más necesitaremos en la práctica.

Ejemplo 2.1.19. Sean a y b los definidos en el Ejemplo 2.1.18 y sean las matrices M y N definidas como sigue:

$$M = \begin{pmatrix} 0 & a \\ b & e \end{pmatrix},\ N = \begin{pmatrix} a & e \\ a & b \end{pmatrix}.$$

Entonces:

$$MN = \begin{pmatrix} 0 & a \\ b & e \end{pmatrix} \begin{pmatrix} a & e \\ a & b \end{pmatrix} = \begin{pmatrix} a^2 & ab \\ ba + a & 2b \end{pmatrix},$$

donde ab y ba son los calculados en (2.6) y (2.7), y donde:

$$\begin{aligned} a^2 &= (1\ 5\ 3) + 4(1\ 3\ 4\ 2) + 2(1\ 4\ 3\ 2) + 4(1\ 4\ 2\ 5) + 2(2\ 3)(4\ 5) + (1\ 2\ 3\ 5\ 4) + 2(1\ 2\ 5\ 4) \\ &\quad + (1\ 4\ 5\ 2\ 3) + 4(2\ 3\ 5), \\ ba + a &= 5(1\ 3\ 2\ 5) + 6(3\ 4\ 5) + 3(1\ 4)(3\ 5) + 3(1\ 5\ 3) + 5(1\ 3\ 4\ 2) + 6(1\ 4\ 3\ 2) + 6(1\ 3\ 4\ 5) \\ &\quad + 3(2\ 4)(3\ 5) + 5(1\ 5\ 3\ 2\ 4) + 6(1\ 3\ 5) + 3(2\ 5\ 3\ 4) + 5(1\ 4)(2\ 5\ 3) \qquad y \\ 2b &= 4(2\ 5) + (1\ 3\ 5) + 2(4\ 5). \end{aligned}$$

2.1.4. Álgebra No Conmutativa

El contenido de esta parte está formado sobre todo por definiciones necesarias para entender el Teorema de Wedderburn-Artin (Teorema 2.1.33) y el de Maschke (Teorema 2.1.34), ambos estudiados en la asignatura de Álgebra No Conmutativa. Debido a la complejidad de algunos de los resultados, no voy a dar demasiados detalles. Los detalles se pueden encontrar en el libro *Associative Algebras* [18].

Definición 2.1.20. Sea R un anillo. Un *R-módulo por la izquierda* es una tripla $(M, +, \cdot)$ en la que $(M, +)$ es un grupo abeliano y $\cdot : R \times M \longrightarrow M$ es una aplicación satisfaciendo:

- $r \cdot (m + n) = r \cdot m + r \cdot n$ para cualesquiera $r \in R$, $m, n \in M$.
- $(r + s) \cdot m = r \cdot m + s \cdot m$ para cualesquiera $r, s \in R$, $m \in M$.
- $r \cdot (s \cdot m) = (r \cdot s) \cdot m$ para cualesquiera $r, s \in R$, $m \in M$.
- $1 \cdot m = m$ para cualquier $m \in M$.

Los *R-módulos por la derecha* se definen de forma similar cambiando la aplicación $\cdot$ y sus propiedades por las análogas por la derecha. Si no hay confusión con el anillo, diremos simplemente que M es un módulo por la izquierda (resp. por la derecha). Diremos que M es un *módulo* si es un módulo por la izquierda y por la derecha.

Ejemplos 2.1.21.

- Todo grupo abeliano es un $\mathbb{Z}$-módulo y viceversa.
- Si $\alpha : A \longrightarrow B$ es un homomorfismo de anillos, entonces todo B-módulo por la izquierda M es un A-módulo por la izquierda definiendo $am := \alpha(a)m$ para todo $a \in A$ y $m \in M$. En ese caso diremos que M es un A-módulo por *restricción de escalares*.

Definición 2.1.22. Sea M un R-módulo. Un subgrupo N de M se dice un *submódulo* si $an \in N$ para cada $a \in R$ y $n \in N$.

Ejemplo 2.1.23. Sea $\alpha : A \longrightarrow B$ un homomorfismo de anillos y sean M y N dos B-módulos con N un submódulo de M. Si ${}_AM$ y ${}_AN$ denotan los A-módulos M y N obtenidos por restricción de escalares, entonces ${}_AN$ es un submódulo de ${}_AM$.

Definición 2.1.24. Sea R un anillo. Diremos que I es un *ideal de R por la izquierda* si es un subgrupo de $(R, +)$ que cumple $RI \subseteq I$. De forma análoga, I será un *ideal de R por la derecha* si es un subgrupo de $(R, +)$ y cumple $IR \subseteq I$. Diremos que I es un *ideal* si es un ideal por la izquierda y por la derecha.

Ejemplo 2.1.25. Si R es un anillo, los submódulos de R por la izquierda son sus ideales por la izquierda.

Definición 2.1.26. Diremos que un módulo M por la izquierda (o por la derecha) es *simple* si sólo tiene dos submódulos. En otras palabras, diremos que el módulo M es simple si $M \neq 0$ y sus únicos submódulos son 0 y M.

Ejemplo 2.1.27. El conjunto de los $\mathbb{Z}$-módulos simples por la derecha es el conjunto vacío.

Definición 2.1.28. Un módulo M es *semisimple* si es suma de submódulos simples. Es decir, si $M = \sum_{i=1}^{n} M_i$ con M_i submódulos simples para todo $i = 1, \ldots, n$.

Al igual que hemos visto los homomorfismos de anillos y de grupos, también existen homomorfismos de módulos.

Definición 2.1.29. Dados dos R-módulos por la izquierda M y N, diremos que la aplicación $f : M \longrightarrow N$ es un *homomorfismo de R-módulos por la izquierda* si es un homomorfismo de grupos verificando $f(am) = af(m)$ para todo $a \in R$ y $m \in M$. Diremos que $f : M \longrightarrow N$ es un *homomorfismo de R-módulos por la derecha* si es un homomorfismo de grupos que cumple $f(ma) = f(m)a$ para todo $a \in R$ y $m \in M$.

Definición 2.1.30. Sea R un anillo conmutativo. Una *R-Álgebra* es un anillo A junto con un homomorfismo de anillos $f : R \longrightarrow Z(A)$ (donde $Z(A) = \{z \in R : zr = rz \ \forall \ r \in R\}$ es el *centro del anillo* A) tal que $ra := f(r)a$ dota a A con la estructura de un R-módulo por la izquierda. Cuando no haya confusión con el anillo diremos que A es un *álgebra*.

Definición 2.1.31. Un *anillo de división* (resp. un *álgebra de división*) es un anillo (resp. un álgebra) con todos sus elementos no nulos invertibles.

Definición 2.1.32. Se dice que un álgebra A es *semisimple por la derecha* (resp. *por la izquierda*) si es semisimple como módulo por la derecha (resp. por la izquierda). Es decir, A es semisimple si $A = \sum_{i=1}^{n} M_i$ y cada M_i es un ideal por la derecha para todo $i = 1, \ldots, n$.

Teorema 2.1.33 (Teorema de Wedderburn-Artin)**.** *Sea A una R-álgebra semisimple por la izquierda o por la derecha. Entonces:*

1. *Existen unos números naturales $n_1, \ldots, n_r$ y unas R-álgebras de división $D_1, \ldots, D_r$ tales que:*

$$A \simeq \mathcal{M}_{n_1}(D_1) \times \ldots \times \mathcal{M}_{n_r}(D_r). \tag{2.8}$$

2. *Los pares $(n_1, D_1), \ldots, (n_r, D_r)$ que satisfacen (2.8) están unívocamente determinados (salvo isomorfismos) por A.*

Demostración. La demostración de este teorema se puede encontrar en el capítulo 3 del libro ya mencionado *Associative Algebras* [18]. □

Si A es una R-álgebra y G es un grupo, entonces el anillo de grupo AG es también una R-álgebra de forma obvia que se suele llamar *álgebra de grupo de G con coeficientes en A*.

Teorema 2.1.34 (Teorema de Maschke)**.** *Sea A un álgebra y sea G un grupo. Se cumple que el álgebra de grupo AG es semisimple si y sólo si A es semisimple, G es finito y $|G| \cdot 1_A$ es invertible en A.*

Demostración. Como la implicación que necesitamos para el trabajo es la de que si A es semisimple, G es finito y $|G| \cdot 1_A$ es invertible entonces AG es semisimple, voy a demostrar sólo esa implicación, la otra se puede encontrar en el capítulo 3 del libro *Associative Algebras* [18].

Por conocimientos de la asignatura de Álgebra No Conmutativa, demostrar que AG es semisimple es equivalente a ver que si tenemos un homomorfismo de AG-módulos por la derecha inyectivo $f : M \longrightarrow N$ entonces existe otro homomorfismo de AG-módulos por la derecha $\overline{\phi} : N \longrightarrow M$ tal que $\overline{\phi} \circ f = 1_M$. En efecto, por ser A semisimple existe un homomorfismo $\phi : N \longrightarrow M$ de A-módulos por la derecha tal que $\phi \circ f = 1_M$. Sea $\overline{\phi} : N \longrightarrow M$ definido por $\overline{\phi}(n) := |G|^{-1} \sum_{g \in G} \phi(ng)g^{-1}$ (tiene sentido poner $|G|^{-1}$ porque sabemos que es invertible por hipótesis). ¿Es esto un homomorfismo de AG-módulos por la derecha que cumple la condición buscada?

Sean $a \in A$ y $h \in G$, entonces:

$$\overline{\phi}(na) = |G|^{-1} \sum_{g \in G} \phi(nag)g^{-1} = |G|^{-1} \sum_{g \in G} \phi(nga)g^{-1} = \left(|G|^{-1} \sum_{g \in G} \phi(ng)g^{-1} \right) a = \overline{\phi}(n)a.$$

$$\begin{aligned} \overline{\phi}(nh) =& |G|^{-1} \sum_{g \in G} \phi(nhg)g^{-1} = |G|^{-1} \sum_{g \in G} \phi(nhg)g^{-1}h^{-1}h = |G|^{-1} \sum_{g \in G} \phi(n(hg))(hg)^{-1} \\ =& |G|^{-1} \sum_{k \in G} \phi(nk)k^{-1}h = \overline{\phi}(n)h. \end{aligned}$$

Con esto ya hemos comprobado que, efectivamente, $\overline{\phi}$ es un homomorfismo de AG-módulos por la derecha (la condición de que se conserve la operación se satisface de forma obvia). Pero, ¿cumple $\overline{\phi} \circ f = 1_M$? Sea $m \in M$:

$$\begin{aligned} (\overline{\phi} \circ f)(m) =& |G|^{-1} \sum_{g \in G} \phi(f(m)g)g^{-1} = |G|^{-1} \sum_{g \in G} (\phi \circ f)(mg)g^{-1} = |G|^{-1} \sum_{g \in G} 1_M(mg)g^{-1} \\ =& |G|^{-1} \sum_{g \in G} mgg^{-1} = |G|^{-1}|G|m = m. \end{aligned}$$

Entonces $\overline{\phi}$ es el homomorfismo que buscábamos, por lo que AG es semisimple como queríamos demostrar. □

Si observamos, todo cuerpo F es semisimple, por lo que tenemos el siguiente corolario.

Corolario 2.1.35. *Sea G un grupo y supongamos que F es un cuerpo. Entonces el álgebra de grupo FG es semisimple si y sólo si G es finito y* $\mathrm{car(F)} \nmid \mathrm{|G|}$.

Observación 2.1.36. Sea el cuerpo $\mathbb{Z}_7$ y sea $\mathcal{S}_5$ el grupo simétrico en 5 elementos. Usando el Corolario 2.1.35 y el Teorema de Wedderburn-Artin 2.1.33, obtenemos que $\mathbb{Z}_7\mathcal{S}_5$ es semisimple y que se puede descomponer en la forma:

$$\mathbb{Z}_7\mathcal{S}_5 \simeq \mathcal{M}_{n_1}(D_1) \times \ldots \times \mathcal{M}_{n_r}(D_r),$$

donde los n_i son números enteros y las D_i son álgebras de división.

2.2. Preliminares de Criptografía

Ya hemos visto en el Capítulo 1 una idea de qué es la Criptografía, pero en esta sección vamos a profundizar en su definición y en algunos de sus aspectos.

Comencemos por lo básico, ¿qué es la Criptografía? Según el libro *A Course in Number Theory and Cryptography* [14], la *Criptografía* es el estudio de métodos de enviar mensajes "disfrazados" de forma que sólo los destinatarios de dichos mensajes puedan ser capaces de "quitar el disfraz a los mensajes" y poder leerlos. Así mismo, el *Criptoanálisis* es el arte de descifrar esos mensajes cifrados. La forma de enviar y descifrar esta información se hace a través de los llamados *criptosistemas*, entre los cuales se pueden distinguir dos tipos: los *criptosistemas simétricos* y los *criptosistemas asimétricos*. Por otro lado, como es habitual en los libros y artículos de Criptografía, para el trabajo necesitaré presentar a tres personajes: Alicia y Bob, que son dos amigos que quieren transmitirse un mensaje secreto entre ellos; y Eva, una espía pasiva que quiere conocer toda la información posible de ese mensaje (la coincidencia de este nombre con el mío es pura coincidencia).

2.2.1. Protocolos criptográficos

Si continuamos usando la definición de criptografía del libro *A Course in Number Theory and Cryptography* [14], llamaremos *mensajes en claro* al conjunto de mensajes que queremos enviar y lo denotaremos por $\mathcal{M}$; y llamaremos *mensajes cifrados* al conjunto de mensajes "disfrazados", denotándolo por $\mathcal{C}$.

La forma de "disfrazar" mensajes y "quitar los disfraces" a los mensajes se hace mediante los *criptosistemas* o *esquemas de cifrado*.

Definición 2.2.1. Un *criptosistema* o *esquema de cifrado* está formado por:

- Dos conjuntos, $\mathcal{K}$ y $\mathcal{K}'$, cuyos elementos llamaremos *claves* o *llaves*.
- Un conjunto $\{c_k : k \in \mathcal{K}\}$ formado por aplicaciones $c_k : \mathcal{M}_k \longrightarrow \mathcal{C}_k$ llamadas *funciones de cifrado* o *transformaciones de cifrado*.
- Un conjunto $\{d_{k'} : k' \in \mathcal{K}'\}$ formado por aplicaciones $d_{k'} : \mathcal{C}_{k'} \longrightarrow \mathcal{M}_{k'}$ llamadas *funciones de descifrado* o *transformaciones de descifrado*.

Estos cuatro conjuntos deben cumplir la condición de que para cada $k \in \mathcal{K}$ debe existir una $k' \in \mathcal{K}'$ tal que $d_{k'} \circ c_k = 1_{\mathcal{M}_k}$. Las claves k y k' son llamadas el *par de claves* y se suele denotar por (k, k'). Nótese que podría darse el caso en el que k y k' fuesen iguales. Usualmente diremos que estamos *cifrando* m cuando estemos calculando $c_k(m)$, y diremos que estamos *descifrando* m cuando estemos calculando $d_{k'}(m)$.

Veamos cómo funciona un esquema de cifrado. Supongamos que Alicia quiere enviar un mensaje m a Bob. Antes de nada, ambos deben tener acordadas las claves k y k' de cifrado y descifrado y, una vez acordadas, Alicia debe calcular $c_k(m) = e$ y enviárselo a Bob. Es en este momento, en el envío y la recepción de la información, cuando podría suceder que Eva interceptara el mensaje cifrado y lo descifrara (si conoce o averigua la clave de descifrado). Una vez el mensaje esté en manos de Bob, éste tendrá que calcular $d_{k'}(e) = d_{k'}(c_k(m)) = m$ y así sabrá qué quería decirle Alicia.

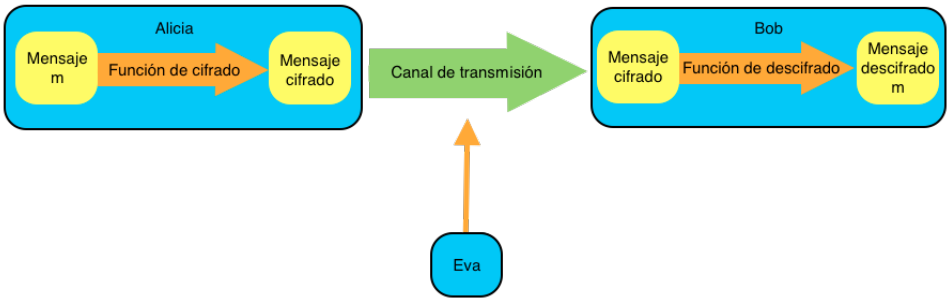
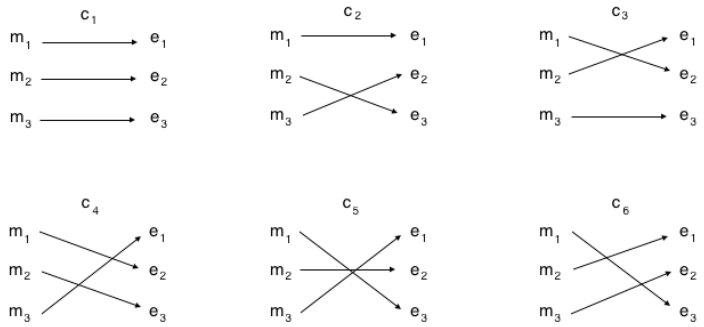


Figura 2.1: Esquema criptográfico.

Ejemplo 2.2.2. Sean $\mathcal{M} = \{m_1, m_2, m_3\}$, $\mathcal{C} = \{e_1, e_2, e_3\}$ y $\mathcal{K} = \mathcal{K}' = \{1, 2, 3, 4, 5, 6\}$. Además, supongamos que las funciones de descifrado cumplen: $d_1 \circ c_2 = 1_{\mathcal{M}}$, $d_2 \circ c_4 = 1_{\mathcal{M}}$, $d_3 \circ c_1 = 1_{\mathcal{M}}$, $d_4 \circ c_3 = 1_{\mathcal{M}}$, $d_5 \circ c_5 = 1_{\mathcal{M}}$ y $d_6 \circ c_6 = 1_{\mathcal{M}}$, siendo las funciones de cifrado de la siguiente forma:



Supongamos que las claves acordadas fuesen $k = 4$ y $k' = 2$. Si el mensaje de Alicia fuera m_3, tendría que calcular $c_4(m_3) = e_1$ y enviar a Bob e_1. Después Bob, para recuperar el mensaje inicial, calcularía $d_2(e_1) = d_2(c_4(m_3)) = m_3$.

Observación 2.2.3. En la práctica, los mensajes que se quieren transmitir son textos escritos en algún alfabeto y que se dividen en porciones de texto más pequeñas. Ni el texto completo ni estas porciones más pequeñas son todavía los mensajes en claro, sino que cada porción de texto en la que se ha dividido el mensaje completo original debe tener asociado uno de los elementos del conjunto de los mensajes en claro de forma biunívoca. El Ejemplo 2.2.2 aclara este comentario.

Una pregunta muy común que se nos podría ocurrir es: ¿por qué es necesario cambiar de clave continuamente?¿No sería más sencillo usar siempre la misma clave? La respuesta a esto es que sí, sería más sencillo usar siempre la misma clave, pero al mismo tiempo sería más inseguro, porque en un esquema criptográfico, los conjuntos $\mathcal{M}$, $\mathcal{C}$, $\mathcal{K}$, $\mathcal{K}'$, $\{c_k : k \in \mathcal{K}\}$ y $\{d_{k'} : k' \in \mathcal{K}'\}$ son públicos y los conoce todo el mundo, mientras que el par (k, k') es privado entre Alicia y Bob. Entonces, si en el esquema criptográfico sólo usaran una clave, nada más sospechar que Eva ha descubierto la clave tendrían que cambiar todo el esquema; sin embargo, si un mismo esquema tiene varias posibles claves, podrían simplemente cambiar la clave y así Eva ya no podría descifrar sus mensajes.

En función de si la clave usada para cifrar el mensaje y la usada para descrifrarlo es la misma o no, podemos distinguir dos tipos de criptosistemas: los simétricos y los asimétricos.

Definición 2.2.4. Un *criptosistema simétrico* es un esquema de cifrado en el que el par de claves (k, k') satisface la condición $k = k'$.

Observación 2.2.5. A los criptosistemas de la Definición 2.2.4 se les llama "simétricos" porque la información necesaria para cifrar un mensaje es esencialmente la misma que se necesita para descifrarlo.

Ejemplos 2.2.6.

1. *Sustitución.* Sea $\mathcal{A}$ un alfabeto con q símbolos y sea $\mathcal{M}$ el conjunto formado por todas las cadenas de longitud t sobre $\mathcal{A}$. Sea $\mathcal{K} = \mathcal{K}'$ el conjunto de todas las permutaciones de los elementos de $\mathcal{A}$. Para cada $\sigma \in \mathcal{K}$, las funciones de cifrado y descifrado son simplemente

$$c_\sigma(m) = (\sigma(m_1)\sigma(m_2)\dots\sigma(m_t)) = (e_1e_2\dots e_t) = e \qquad \text{y}$$

$$d_\sigma(e) = (\sigma^{-1}(e_1)\sigma^{-1}(e_2)\dots\sigma^{-1}(e_t)) = (m_1m_2\dots m_t) = m,$$

donde $m = (m_1m_2\dots m_t) \in \mathcal{M}$ y $e = (e_1e_2\dots e_t) \in \mathcal{C}$.

2. *Reordenamiento.* Sea $\mathcal{A}$ un alfabeto con q símbolos y sea $\mathcal{M}$ el conjunto formado por todas las cadenas de longitud t sobre $\mathcal{A}$. Sea $\mathcal{K} = \mathcal{K}'$ el conjunto de todas las permutaciones de los elementos de $\mathcal{T} = \{1, 2, 3, \ldots, t\}$. Para cada $\sigma \in \mathcal{T}$, las funciones de cifrado y descifrado son

$$c_\sigma(m) = (m_{\sigma(1)} m_{\sigma(2)} \ldots m_{\sigma(t)}) \qquad \text{y} \qquad d_\sigma(e) = (e_{\sigma^{-1}(1)} e_{\sigma^{-1}(2)} \ldots e_{\sigma^{-1}(t)}),$$

donde $m = (m_1 m_2 \ldots m_t) \in \mathcal{M}$ y $e = (e_1 e_2 \ldots e_t) \in \mathcal{C}$.

Prácticamente casi todos los criptosistemas simétricos son combinaciones de los Ejemplos 2.2.6. Sin embargo, hasta ahora no hemos dicho nada sobre las condiciones que deben cumplir las permutaciones, y es en esa elección en la que radica la eficacia o no de un criptosistema simétrico. Lo "bueno o malo" que es un criptosistema vendrá determinado por dos condiciones:

- *La rapidez de los cálculos.* Queremos que sea rápido poder calcular c_k y d_k si se conoce la clave.
- *Seguridad.* Queremos que sea muy lento o imposible (en tiempo) poder calcular c_k y d_k si se desconoce la clave.

Vamos a ver un ejemplo muy conocido de criptosistema simétrico, el *Criptosistema de César*. Este criptosistema ya lo vimos en el Capítulo 1, pero ahora vamos a verlo de una forma "más matemática".

Ejemplo 2.2.7 (Criptosistema de César)**.** Vamos a identificar cada letra del alfabeto español con un entero positivo del 0 al 26, es decir, vamos a identificar nuestro alfabeto con $\mathbb{Z}_{27}$ de acuerdo con la siguiente Tabla 2.3:

A	B	C	D	E	F	G	H	I	J	K	L	M	N
0	1	2	3	4	5	6	7	8	9	10	11	12	13
Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	
14	15	16	17	18	19	20	21	22	23	24	25	26	

Tabla 2.3: Criptosistema de César.

En este criptosistema tenemos $\mathcal{K} = \mathcal{K}' = \mathcal{M} = \mathcal{C} = \mathbb{Z}_{27}$. Además las funciones de cifrado y descifrado son:

$$c_k(m) = m + k \mod 27 \qquad \text{y} \qquad d_k(e) = e - k \mod 27.$$

Por ejemplo, supongamos que la clave es la letra D, es decir, $k = 3$, y que Alicia quiere enviar el mensaje "nos vemos el sábado a las cinco en el acuario".

Lo primero que tiene que hacer Alicia es escribir su mensaje en letras mayúsculas, sin tildes y sin separación entre las palabras, es decir, debe escribir:

NOSVEMOSELSABADOALASCINCOENELACUARIO

Ahora, hay que convertir el mensaje en números usando la Tabla 2.3, obteniendo:

$$13, 15, 19, 22, 4, 12, 15, 19, 4, 11, 19, 0, 1, 0, 3, 15, 0, 11,$$
$$0, 19, 2, 8, 13, 2, 15, 4, 13, 4, 11, 0, 2, 21, 0, 18, 8, 15.$$

Después, Alicia tiene que usar la función de cifrado para obtener:

$$16, 18, 22, 25, 7, 15, 18, 22, 7, 14, 22, 3, 4, 3, 6, 18, 3, 14,$$
$$3, 22, 5, 11, 16, 5, 18, 7, 16, 7, 14, 3, 5, 24, 3, 21, 11, 18.$$

Por último, Alicia tiene que sustituir los números obtenidos por letras y enviar a Bob:

PRVYHORVHÑVDEDGRDÑDVFLPFRHPHÑDFXDULR

Cuando Bob reciba este extraño mensaje, simplemente tendrá que usar la función de descifrado (y separar la frase resultante en palabras que tengan sentido) para obtener el mensaje inicial de "nos vemos el sábado a las cinco en el acuario".

De forma intuitiva, un *criptosistema asimétrico* será aquel en el que k y k' no tengan por qué ser iguales. Esta definición es cierta pero no del todo, en estos criptosistemas cada usuario tendrá dos claves, una pública que los demás usuarios usarán para enviarle mensajes cifrados, y una privada que usará para descifrar dichos mensajes. Sin embargo, la clave privada y la pública no pueden ser dos claves cualesquiera, estarán relacionadas por una aplicación que para que el criptosistema sea seguro debería ser de dirección única. Veamos esto con más detalle.

Definición 2.2.8. Se dice que una función $f : A \longrightarrow B$ es una *función de dirección única* si se verifican las siguientes dos condiciones:

1. Para todo $a \in A$, se puede calcular $f(a)$ en poco tiempo, es decir, en tiempo polinomial con exponente pequeño (la definición de *tiempo de cálculo polinomial* está explicada más adelante en 2.2.19).

2. Para la mayoría de elementos $b \in f(A)$ es difícil encontrar un $a \in A$ tal que $f(a) = b$.

Para aclarar más las ideas vamos a dar una definición alternativa de criptosistema asimétrico.

Definición 2.2.9. Un *criptosistema asimétrico* o de *clave pública* está formado por una función $\mathcal{P} : \mathcal{K}' \longrightarrow \mathcal{K}$ de dirección única, una aplicación $c_k : \mathcal{M}_k \longrightarrow \mathcal{C}_k$ para cada $k \in \mathcal{K}$ y una aplicación $d_{k'} : \mathcal{C}_{k'} \longrightarrow \mathcal{M}_{k'}$ para cada $k' \in \mathcal{K}'$ de manera que si $k = \mathcal{P}(k')$ entonces

$$\mathcal{M}_k = \mathcal{M}_{k'}, \qquad \mathcal{C}_k = \mathcal{M}_{k'}, \qquad \text{y} \qquad d_{k'} \circ c_k = 1_{M_k}$$

El criptosistema de clave pública funciona de la siguiente manera:

- Un usuario elige $k' \in \mathcal{K}'$, que llamaremos su *clave privada*, y calcula $k = \mathcal{P}(k')$, que llamaremos su *clave pública*.
- El usuario hace pública su clave k de forma que todo el mundo pueda enviarle mensajes cifrados usando la función de cifrado $c_k : \mathcal{M}_k \longrightarrow \mathcal{C}_k$.
- Cuando el usuario quiera descifrar un mensaje cifrado $c_k(m)$ que le hayan enviado, deberá usar su clave privada k' y calcular $d_{k'}\left(c_k\left(m\right)\right) = m$.

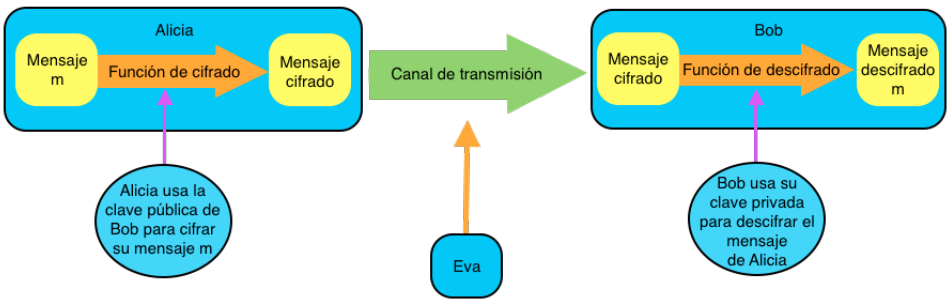


Figura 2.2: Criptografía de clave pública.

De esta manera, una vez cifrado el mensaje de Alicia con la clave pública de Bob, el único que debería poder descifrar el mensaje es Bob con su clave privada. En tal caso se lograría la confidencialidad del mensaje, ya que nadie que no sea el destinatario puede descifrarlo. Visto esto, parece lógico pensar que, puesto que se puede lograr confidencialidad mediante estos criptosistemas, lo mejor es usar siempre criptosistemas de clave pública. Sin embargo, en general los criptosistemas de clave pública son bastante más lentos que los simétricos, así que lo que harán Alicia y Bob habitualmente para enviarse mensajes es usar criptosistemas simétricos; y cada vez que quieran cambiar de clave lo harán usando criptosistemas de clave pública.

2.2.2. Protocolos de intercambio de claves

Los criptosistemas simétricos plantean algunos problemas. El primero de ellos es que la clave se suele cambiar cada cierto tiempo, pero ¿de qué manera puede Alicia decir a Bob cuál es la nueva clave sin que Eva se entere? Lo más sencillo sería pensar que la nueva clave se la pueden enviar en un mensaje usando la antigua clave, pero ¿y si Eva conocía la clave anterior? En ese caso podría conocer la clave nueva al interceptar el mensaje y descifrarlo. El otro problema es: ¿cómo sabe Bob que el mensaje se lo ha enviado Alicia y que no es Eva? Podría suceder que Eva hubiera descubierto la clave para cifrar los mensajes y estuviese tratando de engañar a Bob.

Para solucionar el primer problema, Whitfield Diffie y Martin E. Hellman publicaron en 1976 su artículo *New Directions in Cryptography* [6] en el cual explicaban un método para intercambiar claves, conocido hoy en día como el *Protocolo de Intercambio de Claves de Diffie-Hellman*. De hecho, la idea de clave pública nació con este protocolo de intercambio de claves antes de que la noción de criptosistema de clave pública hubiera aparecido.

Definición 2.2.10. Supongamos que Alicia y Bob quieren ponerse de acuerdo en una nueva clave usando el *Protocolo de Intercambio de Claves de Diffie-Hellman*. Lo primero que tienen que hacer es ponerse de acuerdo en escoger un entero positivo primo p y un elemento g que genere $\mathbb{Z}_p^*$. El conjunto de posibles claves nuevas vendrá dado por $\{[g]_p, [g^2]_p, [g^3]_p, \ldots\}$. En la práctica las claves verdaderas pueden tener cualquier otra naturaleza, pero en este caso se ha fijado una correspondencia biunívoca entre los elementos de $\mathbb{Z}_p^*$ y el conjunto de claves. Además, tanto p como g pueden ser públicos, ya que puede ser que Alicia y Bob estén usando un protocolo criptográfico establecido como un estándar de comunicación (por ejemplo, el protocolo DES [5]). Ahora, los pasos que deben seguir Alicia y Bob son los siguientes:

1. Alicia escoge un entero positivo a, calcula $[g^a]_p$ y hace público el resultado, llamémosle A.
2. Bob elige un entero positivo b, calcula $[g^b]_p$ y hace público el resultado, llamémosle B.
3. Alicia calcula $[B^a]_p$ y esa será su clave k_A.
4. Bob calcula $[A^b]_p$ y esa será su clave k_B.

Después de estos cálculos, Alicia y Bob tienen la misma clave k, ya que:

$$k_A = [B^a]_p \equiv [(g^b)^a]_p \equiv [(g^a)^b]_p \equiv [A^b]_p = k_B.$$

Entonces, en este protocolo, los datos públicos serán p, g, A y B; mientras que a sería un dato privado de Alicia y b sería un dato privado de Bob.

2.2.3. Tiempo de Cálculo

En criptografía, una de las cosas más importantes es que los cálculos usados en los protocolos que cifran nuestros mensajes sean rápidos para las personas que se intercambian la información,

Alicia y Bob; y muy lentos para cualquier persona ajena a ese intercambio y que quiera obtener dicha información sin permiso, Eva. Sin embargo, la velocidad de los cálculos realizados depende de muchos factores tales como el método que se use para realizar el cálculo (el algoritmo), quién realiza las operaciones (una máquina o una persona) y, cómo no, de las prestaciones de la máquina usada (velocidad de cálculo de la máquina, tiempo que tarda en almacenar datos, etc). Por tanto, queremos que nuestro *tiempo de cálculo* sea independiente de todos estos factores, hablando entonces de *tiempo de cálculo de un algoritmo*. Además, lo que más nos importa de ese tiempo de cálculo es cómo se comportará el algoritmo estudiado cuando las entradas sean valores muy grandes, es decir, nos importará el llamado *comportamiento asintótico* del tiempo de cálculo.

La siguiente definición nos servirá para comparar estos comportamientos asintóticos.

Definición 2.2.11. Dadas dos funciones $f, g : \mathbb{N} \longrightarrow \mathbb{R}^+$, diremos que f es una "*o grande*" de g si existe una $C \in \mathbb{R}^+$ tal que $f(n) \leq Cg(n)$ para todo $n \in \mathbb{N}$, es decir, si $\frac{f(n)}{g(n)}$ está acotada superiormente. Esto lo denotaremos por $f = O(g)$.

A continuación estudiaremos el tiempo de cálculo de algunos algoritmos a modo de ejemplo, aunque antes necesitaremos algunas definiciones.

Definición 2.2.12. Diremos que un entero positivo $n \neq 0$ es un *k-dígito en base b* con $b > 0$ si se puede expresar de la forma

$$n = a_{k-1}b^{k-1} + a_{k-2}b^{k-2} + \ldots + a_1 b + a_0,$$

donde $a_{k-1} \neq 0$ y $0 \leq a_i < b$ para todo $i = 0, 1, \ldots, k-1$. Además, definiremos la *longitud de n en base b* como $l_b(n) = k$ y escribiremos el número de la forma $n \equiv (a_{k-1}a_{k-2}\ldots a_1 a_0)_b$ o simplemente $n \equiv a_{k-1}a_{k-2}\ldots a_1 a_0$ cuando la base b esté clara. Si $b = 2$, diremos que n es un *k-bit*.

Observación 2.2.13. Sea $n \neq 0$ un entero positivo. Entonces:

$$l_b(n) = k \qquad \Leftrightarrow \qquad b^{k-1} \leq n < b^k \qquad \Leftrightarrow \qquad k - 1 \leq \log_b(n) < k.$$

Por tanto $\log_b(n) = O(l_b(n))$ y $l_b(n) = k = \lfloor \log_b(n) \rfloor + 1$, donde $\lfloor \log_b(n) \rfloor$ denota el mayor entero menor o igual que el número real $\log_b(n)$. Esto último nos dice que $l_b(n) = O(\log_b(n))$.

Ejemplo 2.2.14 (Tiempo de cálculo de la operación Suma)**.** En este primer ejemplo vamos a calcular cuánto "se tarda" en sumar dos números. Para explicarlo de forma más sencilla vamos a ver un caso concreto, vamos a sumar 108 y 43. Como hemos visto antes, nos interesa que el "tiempo que tarda" sea independiente de las características de la máquina, para lo cual definiremos unas operaciones básicas a las que llamaremos *operaciones bits*. Si expresamos 108 y 43 en base 2 obtenemos:

$$108 \equiv 1101100, \qquad 43 \equiv 101011.$$

Entonces su suma será

$$\begin{array}{cccccccc} & {\scriptstyle 1} & & {\scriptstyle 1} & & & & \\ & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ + & & 1 & 0 & 1 & 0 & 1 & 1 \\ \hline 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}$$

Si nos fijamos, en cada paso del cálculo se realiza una de las siguientes cuatro operaciones elementales que llamaremos *operaciones bit*:

$$0 + 0 + 0 = 0$$
$$1 + 0 + 0 = 1$$
$$1 + 1 + 0 = 10$$
$$1 + 1 + 1 = 11$$

Si consideramos ahora el tiempo necesario para hacer una operación bit como una unidad de tiempo, el tiempo que hemos tardado en realizar la suma de 108 y 43 en binario es de 7 operaciones bits.

Visto esto, podemos generalizar y decir que el tiempo necesario para sumar un k-bit y un s-bit es máx(k, s). Entonces, si n y m son tales que $l_2(n) = k$ y $l_2(m) = s$, obtenemos que el tiempo necesario para sumarlos es aproximadamente $O(\text{máx}(\log_2(n), \log_2(m)))$ por la Observación 2.2.13.

Ejemplo 2.2.15 (Tiempo de cálculo de la operación Multiplicación)**.** Vamos a ver ahora cuánto "se tarda" en multiplicar dos números, en concreto vamos a verlo con los números del ejemplo 2.2.14, es decir, con 108 y 43. Una forma de realizar el producto del ejemplo es con el método que nos enseñan en el colegio, en el cual he suprimido las "llevadas" para que se vea más sencillo:

						1	1	0	1	1	0	0
$\times$							1	0	1	0	1	1
						1	1	0	1	1	0	0
$+$					1	1	0	1	1	0	0	
				1	0	1	0	0	0	1	0	0
$+$			1	1	0	1	1	0	0			
		1	0	0	1	0	1	0	0	1	0	0
$+$	1	1	0	1	1	0	0					
1	0	0	1	0	0	0	1	0	0	1	0	0

Entonces, si queremos multiplicar un k-bit y un s-bit (suponiendo $s \leq k$), y si despreciamos el tiempo usado en mover a la izquierda un número, con este algoritmo necesitaremos usar como máximo $s - 1$ sumas de números con longitud menor o igual que $k + s$. Por tanto, el tiempo necesario para multiplicar estos dos números es $O(s(k + s)) = O(sk) = O(k^2)$ y entonces, si n es un k-bit y m es un s-bit, el tiempo que se tarda en multiplicarlos con este método es, aproximadamente, $O((\log_2(n))^2)$.

Ejemplo 2.2.16 (Método eficaz para calcular potencias)**.** Supongamos que tenemos $g \in G$, con G un semigrupo visto de forma multiplicativa y que queremos saber cuánto es g^a con $a \in \mathbb{N}$. Podemos hacerlo de distinas formas: una de ellas es hacer g, $g \cdot g$, $g^2 \cdot g$, $\ldots$, $g^{a-1} \cdot g = g^a$, lo cual implica hacer $a - 1$ multiplicaciones. Sin embargo, hay otros métodos más eficaces que tardan menos tiempo, uno de ellos es el *método de cuadrar y multiplicar*. Para este algoritmo necesitaríamos un "precálculo" en el cual expresaríamos el exponente a en base 2. Vamos a ver cómo sería ese algoritmo de precálculo y cómo sería el del método de cuadrar y multiplicar:

Algoritmo 1. Expresión en una base.

ENTRADA: $base, num \in \mathbb{Z}^+$; donde num es el número que queremos expresar en la base $base$.

1. $n := num, l := [n \mod base]$.
2. $n = Int(n/base)$, donde Int denota la parte entera de la división.
3. Mientras que $n \neq 0$:
   - 3.1. Añadir $(n \mod base)$ a la lista l,
   - 3.2. $n = Int(n/base)$.

SALIDA: l.

¿Cómo interpretamos el resultado de este algoritmo si por ejemplo le diésemos como entrada $num = 29$ y $base = 2$? En ese caso, el algoritmo nos devolvería la lista $[1, 0, 1, 1, 1]$, es decir, el algoritmo nos diría que el número 29 se puede descomponer de la siguiente forma:

$$29 = 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0.$$

Algoritmo 2. Cuadrar y multiplicar.

ENTRADA: g, a; donde g es un elemento de un semigrupo y $a \in \mathbb{Z}^+$ es la potencia a la que se quiere elevar dicho elemento.

1. $r := 1, potencias := g$.
2. $fact := CBase(2, a)$. Ver 2.2.17 para saber qué es $CBase$.
3. $long :=$ longitud de $fact$.
4. Para $i = 1, \ldots, long$:
   - 4.1. Si la posición i de $fact$ tiene un 1, entonces $r = r \cdot potencias$.
   - 4.2. $potencias = potencias \cdot potencias$.

SALIDA: r.

Vamos a ver cómo funcionaría este último algoritmo para calcular g^{29}. Como hemos visto antes, el algoritmo de cambio de base nos indica que $29 = 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$. Entonces, el algoritmo de cuadrar y multiplicar haría el siguiente cálculo:

$$g \cdot \left(g^2\right)^2 \cdot \left(\left(g^2\right)^2\right)^2 \cdot \left(\left(\left(g^2\right)^2\right)^2\right)^2,$$

para el cual sólo hay que hacer como mucho k multiplicaciones (suponiendo que a es un k-bit), es decir, aproximadamente $O(\log_2(a))$ multiplicaciones, un tiempo mucho menor que el número de multiplicaciones que tendríamos que hacer en el método usual.

Observación 2.2.17. Todos los algoritmos que van a aparecer a lo largo del trabajo los he programado con el software GAP [9]. Concretamente, si nos fijamos en el Ejemplo 2.2.16, al algoritmo que cambia de base lo he llamado *CBase*, mientras que al de cuadrar y multiplicar lo he llamado *Potencia*.

2.2.4. Problemas y algoritmos

Definición 2.2.18.

- Un *problema* es una descripción general de una tarea que depende de unos parámetros. Además, se define una *especificación de un problema* como el problema que se obtiene al darle valor a los parámetros.
- Un *algoritmo* es una lista de instrucciones que sirven para resolver un problema. Las posibles entradas del algoritmo son las diferentes especificaciones del problema, y el algoritmo debe tener una salida.
- Se dice que un problema es de *decisión* si el conjunto formado por las posibles salidas del problema es $\mathcal{B} = \{\text{Verdadero, Falso}\}$.

Como es lógico, si tenemos un problema nos gustaría que el algoritmo que lo resuelve no tarde demasiado tiempo en darnos el resultado, esto se traduce en que queremos que el algoritmo sea de *tiempo polinomial*, cuya definición vamos a ver a continuación.

Definición 2.2.19. Se dice que un algoritmo es de *tiempo de cálculo polinomial* o de *tiempo polinomial* si existe $d \in \mathbb{N}$ tal que el tiempo de cálculo del algoritmo para una especificación de longitud k es $O(k^d)$.

Si usamos ahora la Observación 2.2.13 obtenemos que si la especificación es un número n escrito en una base b, entonces que el algoritmo tenga tiempo polinomial es equivalente a que el tiempo de cálculo del algoritmo sea $O((\log_b(n))^d)$ para un cierto $d \in \mathbb{N}$.

Ejemplo 2.2.20. Si nos fijamos en el Protocolo de Intercambio de Claves de Diffie-Hellman visto en 2.2.10, parece que Alicia y Bob tienen que hacer muchas operaciones para calcular las claves. Sin embargo, si están por ejemplo en el grupo $\mathbb{Z}$, el tiempo empleado en calcular la potencia g^a (siendo a un k-bit) mediante el algoritmo de *cuadrar y multiplicar* del Ejemplo 2.2.16, sus cuentas tendrán un tiempo de cálculo polinomial, puesto que el tiempo empleado será aproximadamente $O((\log_2(a))^3)$.

Definición 2.2.21. Los algoritmos que hemos visto hasta ahora han sido *deterministas*, es decir, algoritmos que si los ejecutas varias veces con la misma entrada realizan los mismos cálculos cada vez y te devuelven el mismo resultado. Por el contrario, los *algoritmos probabilísticos* realizan decisiones aleatorias en alguno de los pasos del algoritmo, pudiendo suceder que dos entradas iguales den resultados distintos.

Como los algoritmos vistos hasta ahora han sido deterministas no es necesario poner ejemplos de este tipo de algoritmos. Sin embargo, como ese no es el caso para los algoritmos probabilísticos vamos a ver el siguiente ejemplo.

Ejemplo 2.2.22. Supongamos que dado un número entero x queremos saber si es compuesto. Un posible algoritmo puede ser el siguiente:

1. Tomamos un entero y al azar y calculamos $\text{mcd}(\text{x}, \text{y})$.

2. Si $\mathrm{mcd}(\mathrm{x}, \mathrm{y}) \neq 1$ y $\mathrm{mcd}(\mathrm{x}, \mathrm{y}) \neq \mathrm{x}$ entonces el número x es compuesto y esa sería la salida del algoritmo.

3. Si $\mathrm{mcd}(\mathrm{x}, \mathrm{y}) = 1$ o $\mathrm{mcd}(\mathrm{x}, \mathrm{y}) = \mathrm{x}$ no sabemos nada y la salida del algoritmo podría ser que el método ha fallado.

En el caso de que el algoritmo nos diera como salida un fallo, lo que se haría en la práctica no sería pararse en el primer entero elegido al azar, sino volver al paso 1. Si el número x es compuesto esperamos encontrar, después de un número suficiente de pasos del bucle, un entero y para el que el algoritmo pare en el paso 2. Si x es primo, esto no pasará nunca, pero con este método no podemos asegurar en ningún momento que x lo sea. Sin embargo, si ya hemos probado con una gran cantidad de números y con todos hemos llegado a que el máximo común divisor es 1 o x, podemos decir que el número no es compuesto con cierta probabilidad.

Este algoritmo no es el mejor método ni el más eficiente, porque podría suceder que a pesar de haber probado con muchos números no hayamos encontrado el correcto y pensemos que el número no es compuesto a pesar de serlo. Sin embargo, la elección aleatoria del entero y muestra perfectamente el porqué de la palabra "probabilístico".

Capítulo 3

El Problema del Logaritmo Discreto

Uno de los problemas más importantes en la Criptografía es el llamado *Problema del Logaritmo Discreto*. En este capítulo voy a usar los contenidos de las referencias [16], [19], [20] y [23].

3.1. Definición del problema

Definición 3.1.1. Sea S un semigrupo finito. Se define el *Problema del Logaritmo Discreto* como el problema en el que dados $a, b \in S$ se quiere encontrar un $x \in \mathbb{N}$ tal que

$$a^x = b. \tag{3.1}$$

Obviamente, para que el problema tenga solución, b debe estar en el semigrupo generado por a.

La dificultad o no de resolver este problema varía mucho dependiendo del semigrupo S escogido. Por ejemplo, si S es $\mathbb{Z}_n$, la ecuación (3.1) se transforma en una ecuación de congruencias de la forma $ax \equiv b \mod n$ y que sabemos resolver sin problemas usando el Algoritmo de Euclides y la Identidad de Bézout explicados en la Sección 2.1. Vamos a verlo con unos ejemplos.

Ejemplo 3.1.2. Supongamos que tenemos que resolver la ecuación

$$4x \equiv 3 \mod 7.$$

Como 4 y 7 son coprimos, sabemos que existe un inverso para el número 4 en $\mathbb{Z}_7$ por la Proposición 2.1.3. Además, rápidamente nos damos cuenta de que $[4]_7 \cdot [2]_7 = [1]_7$, luego multiplicamos ambos miembros de la ecuación por 2 y obtenemos

$$x \equiv 2 \cdot 3 = 6 \mod 7.$$

Por tanto, el número buscado es $x = 6$ (aunque también será solución cualquier número que sea de la forma $7\lambda + 6$).

Ejemplo 3.1.3. Supongamos que tenemos que resolver

$$77x \equiv 30 \mod 180. \tag{3.2}$$

De nuevo, por la Proposición 2.1.3, como 77 y 180 son coprimos, sabemos que existe un inverso para 77 en $\mathbb{Z}_{180}$. Sin embargo, en este caso no es tan sencillo calcularlo y tendremos que recurrir al Algoritmo de Euclides y a la Identidad de Bézout. La tabla resultante del Algoritmo de Euclides es la siguiente:

	2	2	1	25
180	77	26	25	1
26	25	1	0	

Tabla 3.1: Algoritmo de Euclides con 180 y 77.

Así que $\text{mcd}(180, 77) = 1$. Si ahora nos fijamos en las divisiones efectuadas en la Tabla 3.1

$$\begin{aligned} 180 &= 77 \cdot 2 + 26 \\ 77 &= 26 \cdot 2 + 25 \\ 26 &= 25 \cdot 1 + 1 \\ 25 &= 1 \cdot 25 \end{aligned}$$

y vamos despejando y sustituyendo, obtenemos

$$\begin{aligned} 1 &= 26 - 25 = 26 - (77 - 26 \cdot 2) \\ &= -77 + 26 \cdot 3 = -77 + (180 - 77 \cdot 2) \cdot 3 \\ &= 180 \cdot 3 + 77 \cdot (-7). \end{aligned}$$

Por tanto, una Identidad de Bézout para 180 y 77 es $1 = 180 \cdot 3 + 77 \cdot (-7)$, luego

$$-7 \cdot 77 \equiv 1 \quad \text{mód } 180.$$

Entonces, multiplicando por -7 en los dos miembros de la ecuación (3.2), llegamos a que

$$x \equiv -7 \cdot 30 = -210 \equiv -30 \quad \text{mód } 180.$$

Así que el número que buscábamos era $x = -30$ (o cualquier número de la forma $180\lambda - 30$).

Queda claro con estos ejemplos que, si estamos en $\mathbb{Z}_n$, el problema es bastante sencillo de resolver, porque además, el Algoritmo de Euclides se puede programar fácilmente en cualquier ordenador. Sin embargo, ¿qué podemos hacer si el semigrupo S es otro distinto de $\mathbb{Z}_n$? Para esos casos existen varios tipos de ataques, de entre los cuales vamos a ver los más comunes y que vamos a necesitar para entender algunos de los argumentos que dan D. Kahrobael, C. Koupparis y V. Shpilrain para justificar la validez de su método en el artículo *Public Key Exchange Using Matrices Over Group Rings* [13].

3.2. Fuerza Bruta

Este método consiste en, dada la ecuación (3.1), probar con distintos valores de x hasta encontrar el que resuelve la ecuación. Como se puede suponer, este método no es muy efectivo porque precisamente para dificultar los cálculos de este método se coge siempre un elemento $a \in S$ tal que su orden sea aproximadamente 10^{300}.

Para explicar este método vamos a ver un ejemplo en el que, para poder realizarlo "a mano", S va a ser relativamente pequeño.

Ejemplo 3.2.1. Supongamos que estamos en $\mathbb{Z}_{89}^*$ y que queremos encontrar un entero x tal que $5^x \equiv 8 \mod 89$ mediante el método de la fuerza bruta (el orden de 5 en $\mathbb{Z}_{89}^*$ es 44). Si calculamos las sucesivas potencias hasta encontrar un x que satisfaga la ecuación, nos quedará una tabla como la que sigue:

i	0	1	2	3	4	5	6	7	8	9	10	11	12
5^i mód 89	1	5	25	36	2	10	50	72	4	20	11	55	8

Tabla 3.2: Fuerza bruta.

Observando la Tabla 3.2 podemos observar que $x = 12$ cumple la condición que queríamos.

3.3. Algoritmo de Shanks: Paso de Niño-Paso de Gigante

El siguiente algoritmo, debido a Daniel Shanks (1917-1996) y cuyo nombre en inglés es "Baby Step-Giant Step", es uno de los más conocidos, y recibe su nombre por la forma que tiene el algoritmo. Este algoritmo no sirve si estamos sólo en un semigrupo puesto que necesitamos en cierto momento calcular un inverso. Por tanto, necesitaremos la condición más fuerte de estar en un grupo G.

Algoritmo 3. Paso de Niño-Paso de Gigante.

ENTRADA: a, b y n con a y b elementos de un grupo G y n una cota superior del orden de a en G.

1. $m := \lceil \sqrt{n} \rceil = \text{mín}\{k \in \mathbb{Z} : \sqrt{n} \leq k\}$.
2. Para $i = 0, \ldots, m-1$:
   2.1. Calcular y guardar (i, ba^{-i}).
3. Para $j = 0, \ldots, m-1$:
   3.1. Calcular $a_j = a^{jm}$.
   3.2. Si $a_j = ba^{-i}$ para algún i, SALIDA: $i + jm$.
4. Si el algoritmo no se acaba en el bucle anterior entonces:
   4.1. SALIDA: "b no está en el grupo generado por a".

La primera parte del algoritmo, en la que se calculan los ba^{-i}, es la que se conoce como "Paso de Niño", pues el exponente i se va aumentando de uno en uno. El "Paso de Gigante" es la segunda parte, cuando calculamos los a^{jm}, ya que el exponente de a se va aumentando con múltiplos de m.

¿Por qué funciona este método cuando b está en el grupo generado por a? Es decir, ¿por qué hay un elemento en la tabla de pasos de niño que coincide con un elemento de los calculados con paso de gigante? Como $m := \lceil \sqrt{n} \rceil$, entonces $m^2 > n$ y podemos asumir que la solución de $a^x = b$ cumple $0 \leq x < m^2$. Entonces, sabemos que x se puede descomponer en la forma $x = x_0 + x_1 m$, donde $x_0 \equiv x \mod m$ y $x_1 = (x - x_0)/m$. Además, en dicha descomposición tenemos que $0 \leq x_0, x_1 < m$. Por tanto, si en el algoritmo tenemos $i = x_0$ y $j = x_1$, sucederá que $ba^{-i} = ba^{-x_0} = a^x a^{-x_0} = a^{x-x_0} = a^{x_1 m} = a^{jm}$, luego existe una coincidencia en las listas. Sin embargo, a pesar de parecer tan sencillo, como mínimo tendremos que hacer $\sqrt{n}$ multiplicaciones, lo cual hace que no sea uno de los algoritmos más rápidos o eficientes.

Ejemplo 3.3.1. Supongamos que estamos en $\mathbb{Z}_{131}^*$ y que queremos encontrar un entero x tal que $7^x \equiv 123 \mod 131$. Como 131 es primo, sabemos que $\mathbb{Z}_{131}^*$ tiene 130 elementos, así que tomamos $n = 130$ como cota superior del orden de 7 en $\mathbb{Z}_{131}^*$, y por tanto $m = \lceil \sqrt{130} \rceil = 12$.

La tabla de los pasos de niño es la siguiente:

i	0	1	2	3	4	5	6	7	8	9	10	11
$123 \cdot 7^{-i} \mod 131$	123	55	64	84	12	114	35	5	113	91	13	58

Tabla 3.3: Pasos de niño.

Los pasos de gigante son:

j	0	1	2	3	4	5
$7^{12j} \mod 131$	1	121	100	48	44	84

Tabla 3.4: Pasos de gigante.

Como el número correspondiente a $j = 5$ aparece en la Tabla 3.3, ya no hace falta calcular más pasos de gigante, y el algoritmo nos dice que el número buscado x es $x = i + jm = 3 + 5 \cdot 12 = 63$.

Observación 3.3.2. Para calcular las potencias del tipo ba^{-i}, en concreto para calcular a^{-i}, lo que hay que hacer es calcular el inverso de a en G con el Algoritmo de Euclides como hicimos en el Ejemplo 3.1.3 y luego elevarlo a i de la forma habitual.

3.4. Algoritmo de Pohlig-Hellman

Este algoritmo se basa en el Teorema Chino de los Restos 2.1.4. La idea es la siguiente: supongamos que queremos, de nuevo, encontrar un entero x tal que $a^x = b$ con a y b elementos de un semigrupo S. Supongamos que el orden de a es n y que conocemos una factorización suya en primos de la forma

$$n = \prod_{i=1}^{r} p_i^{e_i}.$$

Entonces se pueden encontrar $x_i \equiv x \mod p_i^{e_i}$ para cada $i = 1, \ldots, r$ y después, usando el Teorema Chino de los Restos, encontraremos la solución. Además, cada entero x_i se podrá descomponer en la forma $x_i = l_0 + l_1 p_i + l_2 p_i^2 + l_{e_i - 1} p_i^{e_i - 1}$ con los l_j cumpliendo $0 \leq l_j \leq p_i - 1$.

Algoritmo 4. Pohlig - Hellman.

ENTRADA: a, b y n con a y b elementos de un semigrupo S y a generando el semigrupo S de orden n.

1. Encontrar la factorización en primos de n: $n = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r}$, donde los $e_i \geq 1$.
2. Para $i = 1, \ldots, r$:
   - 2.1. Para simplificar, definimos $q = p_i$ y $e = e_i$.
   - 2.2. $\overline{a} = a^{n/q}$.
   - 2.3. $\gamma = 1$, $l_{-1} = 0$.
   - 2.4. Para $j = 0, \ldots, e-1$:
     - 2.4.1. $\gamma = \gamma a^{l_{j-1} q^{j-1}}$, $\overline{b} = (b\gamma^{-1})^{n/q^{j+1}}$.
     - 2.4.2. $l_j = \log_{\overline{a}} \overline{b}$.
   - 2.5. $x_i = l_0 + l_1 q + \ldots + l_{e-1} q^{e-1}$.
3. Usar el Teorema Chino de los Restos para obtener el x, con $0 \leq x \leq n-1$), tal que $x \equiv x_i \mod p_i^{e_i}$ para $1 \leq i \leq r$.

SALIDA: x.

¿Por qué funciona este método? ¿Los l_j están bien definidos de la forma en la que lo hace el algoritmo? Veámoslo. Primero, observemos que como el orden de a es n, entonces el orden $\overline{a}$ en el paso 2.3 es q. Ahora, si nos fijamos, en una iteración k del paso 2.4 tendremos $\gamma = a^{l_0 + l_1 q + l_2 q^2 + \ldots + l_{k-1} q^{k-1}}$. Por tanto,

$$\begin{aligned} \overline{b} &= (b/\gamma)^{n/q^{k+1}} = (a^{x - l_0 - l_1 q - l_2 q^2 - \ldots - l_{k-1} q^{k-1}})^{n/q^{k+1}} \\ &= (a^{n/q^{k+1}})^{x - l_0 - l_1 q - l_2 q^2 - \ldots - l_{k-1} q^{k-1}} \equiv (a^{n/q^{k+1}})^{x_i - l_0 - l_1 q - l_2 q^2 - \ldots - l_{k-1} q^{k-1}} \\ &= (a^{n/q^{k+1}})^{l_k q^k + \ldots + l_{e-1} q^{e-1}} = (a^{n/q})^{l_k + \ldots + l_{e-1} q^{e-1-k}} = (\overline{a})^{l_j} \mod q^e. \end{aligned} \tag{3.3}$$

Viendo las operaciones (3.3), queda claro que $l_j = \log_{\overline{a}} \overline{b}$ tal y como hacemos en el algoritmo.

Ejemplo 3.4.1. Supongamos que estamos en $\mathbb{Z}_{601}^*$ y que queremos encontrar un entero x tal que $19^x \equiv 329 \mod 601$. Sabemos que $\mathbb{Z}_{601}^*$ es de orden 600. Por otro lado, puede probarse (aunque no es el objetivo de este ejemplo) que el orden de 19 en el grupo $\mathbb{Z}_{601}^*$ es 600, por lo que 19 es un elemento generador de $\mathbb{Z}_{601}^*$. Vamos a realizar los pasos del algoritmo con $a = 19$, $b = 329$ y $n = 600$.

El primer paso es factorizar el número 600. Es decir:

$$600 = 2^3 \cdot 3 \cdot 5^2,$$

de donde, usando la notación del algoritmo, obtenemos que $p_1 = 2$, $e_1 = 3$, $p_2 = 3$, $e_2 = 1$, $p_3 = 5$ y $e_3 = 2$.

Vamos con el paso 2. En este caso, la i va a ir desde 1 hasta 3.

1. Cuando $i = 1$, tenemos $q = 2$, $e = 3$ y $\overline{a} = 19^{600/2} \equiv 600 \mod 601$. Por otro lado, cuando hagamos el bucle con j, sus valores serán 0, 1 y 2.

- $\gamma = 1$, $l_{-1} = 0$.
- Si $j = 0$, entonces:

$$\gamma = 1 \cdot 19^0 \equiv 1 \mod 601 \qquad \text{y} \qquad \overline{b} = (329 \cdot 1^{-1})^{600/2} \equiv 600 \mod 601.$$

  Por tanto $l_0 = \log_{600} 600 = 1$.
- Si $j = 1$:

$$\gamma = 1 \cdot 19^{1 \cdot 2^0} \equiv 19 \mod 601 \qquad \text{y} \qquad \overline{b} = (329 \cdot 19^{-1})^{600/2^2} \equiv 1 \mod 601.$$

  Entonces $l_1 = \log_{600} 1 = 0$.
- Si $j = 2$:

$$\gamma = 19 \cdot 19^0 \equiv 19 \mod 601 \qquad \text{y} \qquad \overline{b} = (329 \cdot 19^{-1})^{600/2^3} \equiv 1 \mod 601.$$

  Ahora $l_2 = \log_{600} 1 = 0$.
- Entonces obtenemos que $x_1 = 1 + 0 \cdot 2 + 0 \cdot 2^2 = 1$.

2. Cuando $i = 2$, tenemos $q = 3$, $e = 1$ y $\overline{a} = 19^{600/3} \equiv 576 \mod 601$. En este caso no va a haber bucle, ya que j varía de 0 a $e - 1 = 1 - 1 = 0$.
   - $\gamma = 1$, $l_{-1} = 0$.
   - Con $j = 0$ tenemos:

$$\gamma = 1 \cdot 19^0 \equiv 1 \mod 601 \qquad \text{y} \qquad \overline{b} = (329 \cdot 1^{-1})^{600/3} \equiv 24 \mod 601.$$

     Por tanto $l_0 = \log_{576} 24 = 2$.
   - Así que $x_2 = 2$.
3. Por último, cuando $i = 3$, tenemos $q = 5$, $e = 2$ y $\overline{a} = 19^{600/5} \equiv 423 \mod 601$. Esta vez, los valores de j serán 0 y 1.
   - $\gamma = 1$, $l_{-1} = 0$.
   - Cuando $j = 0$:

$$\gamma = 1 \cdot 19^0 \equiv 1 \mod 601 \qquad \text{y} \qquad \overline{b} = (329 \cdot 1^{-1})^{600/5} \equiv 32 \mod 601.$$

     Así que $l_0 = \log_{423} 32 = 3$.
   - Cuando $j = 1$:

$$\gamma = 1 \cdot 19^{3 \cdot 5^0} \equiv 248 \mod 601 \qquad \text{y} \qquad \overline{b} = (329 \cdot 248^{-1})^{600/5^2} \equiv 32 \mod 601.$$

     Entonces $l_1 = \log_{423} 32 = 3$.
   - Por tanto, $x_3 = 3 + 3 \cdot 5 = 18$.

Entonces, hemos llegado a un sistema de ecuaciones de congruencias

$$\begin{cases} x \equiv 1 \mod 8 \\ x \equiv 2 \mod 3 \\ x \equiv 18 \mod 25 \end{cases}$$

que podemos resolver mediante el Teorema Chino de los Restos 2.1.4.

Sean $M_1 = 600/8 = 75$, $M_2 = 600/3 = 200$ y $M_3 = 600/25 = 24$. Debemos calcular N_1, N_2 y N_3 tales que:

$$M_1N_1 \equiv 1 \mod 8, \qquad M_2N_2 \equiv 1 \mod 3 \qquad \text{y} \qquad M_3N_3 \equiv 1 \mod 25.$$

Es decir, queremos resolver:

$$75N_1 \equiv 1 \mod 8, \qquad 200N_2 \equiv 1 \mod 3 \qquad \text{y} \qquad 24N_3 \equiv 1 \mod 25.$$

Fácilmente y como hemos hecho en ejemplos anteriores podemos obtener los inversos que buscamos usando el Algoritmo de Euclides y la Identidad de Bézout. En este caso:

$$N_1 = 3, \qquad N_2 = -1 \qquad \text{y} \qquad N_3 = -1.$$

Por tanto, la solución x que buscábamos es:

$$x = 1 \cdot 75 \cdot 3 + 2 \cdot 200 \cdot (-1) + 18 \cdot 24 \cdot (-1) = -607 \equiv 593 \mod 600.$$

Es decir, $19^{593} \equiv 329 \mod 601$ como queríamos.

3.5. Algoritmo ρ de Pollard para logaritmos

Este algoritmo consiste en construir una secuencia pseudoaleatoria de elementos de G, con G un grupo finito, en la que existan dos elementos iguales y, a partir de ellos, poder resolver nuestro Problema 3.1.1 de dados $a, b \in G$ encontrar x tal que $a^x = b$. En este caso necesitaremos que a tenga orden n con n primo. Vamos a explicar algunos detalles para poder entender el algoritmo.

Primero, tomaremos una partición de G en tres conjuntos S_1, S_2 y S_3 de aproximadamente el mismo tamaño con la condición de que $1 \notin S_2$. La secuencia antes comentada vendrá dada por:

$$x_0 = 1, \qquad x_{i+1} = f(x_i) = \begin{cases} b \cdot x_i & \text{si } x_i \in S_1 \\ x_i^2 & \text{si } x_i \in S_2 \\ a \cdot x_i & \text{si } x_i \in S_3 \end{cases} \tag{3.4}$$

para $i \geq 0$.

Como $a^x = b$, el sistema (3.4) nos dice que para todo $i \geq 0$ se tiene $x_i = a^{\alpha_i} b^{\beta_i}$ donde los enteros α_i y β_i pueden calcularse de la siguiente forma:

$$\alpha_0 = 0, \qquad \alpha_{i+1} = \begin{cases} \alpha_i & \text{si } x_i \in S_1 \\ 2 \cdot \alpha_i \mod n & \text{si } x_i \in S_2 \\ \alpha_i + 1 \mod n & \text{si } x_i \in S_3 \end{cases} \tag{3.5}$$

$$\beta_0 = 0, \qquad \beta_{i+1} = \begin{cases} \beta_i + 1 \mod n & \text{si } x_i \in S_1 \\ 2 \cdot \beta_i \mod n & \text{si } x_i \in S_2 \\ \beta_i & \text{si } x_i \in S_3 \end{cases} \tag{3.6}$$

Ahora, se trata de encontrar dos enteros i,j tales que $x_i = x_j$, para así obtener:

$$a^{\alpha_i} b^{\beta_i} = a^{\alpha_j} b^{\beta_j},$$

lo cual es equivalente, usando que $a^x = b$ a:

$$a^{\alpha_i} a^{x\beta_i} = a^{\alpha_j} a^{x\beta_j}.$$

Esto último es lo mismo que

$$a^{x(\beta_i-\beta_j)} = a^{\alpha_j-\alpha_i}.$$

Si ahora tomamos logaritmo en base a en los dos miembros llegamos a

$$x(\beta_i - \beta_j) \equiv \alpha_j - \alpha_i \mod n.$$

Denotando $r := \beta_i - \beta_j$ y $u := \alpha_j - \alpha_i$, nuestro problema de resolver un logaritmo discreto se ha reducido a un problema mucho más sencillo: resolver una ecuación de congruencias $xr \equiv u$ mód n. Una vez resuelta la ecuación de congruencias habremos obtenido nuestro x buscado. Esta ecuación tendrá sentido si $r \neq 0$, ¿qué hacemos si nos sale $r = 0$? En ese caso deberemos repetir el proceso tomando tomando un $x_0 \neq 1$. Sin embargo, según los autores del libro *Handbook of Applied Cryptography* [16], el suceso de que r sea cero se da con una probabilidad despreciable.

Al ver esto nos surge otra pregunta, ¿cómo encontramos la i y la j? Cuando Pollard publicó en 1978 su artículo *Monte Carlo Methods for Index Computation mod p* [19], solventó este problema usando la *Detección de Ciclos de Floyd*, que nos dice que una forma de encontrar los ciclos es ir calculando las parejas x_i y x_{2i} con $i \geq 0$. Esta idea es la que se usa en el algoritmo.

Vamos a ver en qué consiste la Detección de Ciclos de Floyd antes de ponernos a ver el algoritmo de Pollard.

Definición 3.5.1. Se dice que la sucesión $\{x_i\}_{i\in\mathbb{N}}$ tiene un *ciclo* de longitud λ, con λ un entero, si existe un entero μ tal que para todo índice j mayor o igual que μ se tiene que $x_j = x_{j+\lambda}$. En ese caso, se verifica además que $x_j = x_{j+k\lambda}$ para todo entero $k \geq 0$.

Si representamos gráficamente la situación que se da en la Definición 3.5.1 vemos que obtenemos una forma parecida a la letra ρ griega, y debido a esto, al método que estamos estudiando se le llama "algoritmo ρ" de Pollard.

Figura 3.1: Ciclo en una sucesión. Fuente: [12]

A no ser que se especifique lo contrario, consideraremos que λ y μ son los menores enteros positivos que verifican las condiciones de la Definición 3.5.1 de ciclo. Además, si observamos, cuanto menores sean ambos valores, más rápido encontraremos el bucle y más rápido será el método.

Teorema 3.5.2 (Detección de Ciclos de Floyd)**.** *Si en una sucesión $\{x_i\}_{i\in\mathbb{N}}$ existe un ciclo, entonces existe un número natural i tal que $x_i = x_{2i}$. Además,el menor i tal que $x_i = x_{2i}$ cumple que $\mu \leq i \leq \mu + \lambda$, donde μ es el índice del primer elemento del ciclo y λ es la longitud del ciclo.*

Demostración. Por la Definición 3.5.1 de ciclo, existen λ y μ tales que para todo entero positivo $j \geq \mu$ se cumple $x_j = x_{j+k\lambda}$ (para todo entero positivo k). Por tanto, si $i = k\lambda \geq \mu$ tendremos que $x_i = x_{2i}$. Veamos ahora la cota para el menor i que cumple la igualdad. Está claro que

$i \geq \mu$ puesto que, en caso contrario, x_i no estaría en el ciclo. Por otro lado, supongamos que i fuese mayor estricto que $\mu + \lambda$, es decir, $i = \mu + \lambda + s$ con $s \gneq 0$ en ese caso:

$$x_i = x_{\mu+\lambda+s} = x_{\mu+s} \qquad \text{y} \qquad x_{2i} = x_{2(\mu+\lambda+s)} = x_{2(\mu+s)}.$$

Pero entonces, usando que $x_i = x_{2i}$ tenemos que $x_{\mu+s} = x_{2(\mu+s)}$ con $\mu + s < \mu + s + \lambda = i$, contradiciendo el hecho de que i sea el mínimo cumpliendo la condición. □

Vista la Detección de Ciclos de Floyd, ya estamos en disposición de entender el *algoritmo ρ de Pollard*.

Algoritmo 5. ρ de Pollard para logaritmos.

ENTRADA: a, b y n con a y b elementos de un grupo G y a generando el grupo G de orden n.

1. $x_0 = 1$, $\alpha_0 = 0$, $\beta_0 = 0$.
2. Para $i = 1, 2, \ldots$:
   - 2.1. Usando los valores x_{i-1}, α_{i-1}, β_{i-1}, x_{2i-2}, α_{2i-2} y β_{2i-2} calculados anteriormente y las ecuaciones 3.4, 3.5 y 3.6, calcular x_i, α_i, β_i, x_{2i}, α_{2i} y β_{2i}.
   - 2.2. Si $x_i = x_{2i}$ entonces $r = \beta_i - \beta_{2i} \mod n$.
   - 2.3. Si $r = 0$, entonces SALIDA: "$r = 0$ y no se puede hacer el algoritmo, hay que cambiar el valor de x_0".
   - 2.4. Si $r \neq 0$, entonces calcular $x = r^{-1}(\alpha_{2i} - \alpha_i) \mod n$. SALIDA: x.

Veamos un ejemplo de cómo aplicar este algoritmo.

Ejemplo 3.5.3. Supongamos que estamos en $\mathbb{Z}_{71}^*$ y que queremos encontrar un entero x tal que $7^x \equiv 59 \mod 71$. Sabemos que $\mathbb{Z}_{71}^*$ es de orden 70. Además, puede probarse (aunque no es el objetivo de este ejemplo) que 7 módulo 71 genera el grupo $\mathbb{Z}_{71}^*$. Vamos pues, a realizar el algoritmo ρ de Pollard para logaritmos con $a = 7$, $b = 59$ y $n = 70$.

La partición que voy a hacer va a ser la siguiente:

$$S_1 = \{g \in \mathbb{F}_{71}^* : g \equiv 1 \mod 3\},$$

$$S_2 = \{g \in \mathbb{F}_{71}^* : g \equiv 0 \mod 3\},$$

$$S_3 = \{g \in \mathbb{F}_{71}^* : g \equiv 2 \mod 3\}.$$

Ahora, si hacemos las operaciones dadas por las ecuaciones (3.4), (3.5) y (3.6), obtenemos una tabla como la siguiente.

i	x_i	α_i	β_i	$2i$	x_{2i}	α_{2i}	β_{2i}
1	59	0	1	2	58	1	1
2	58	1	1	4	27	2	2
3	14	1	2	6	56	4	5
4	27	2	2	8	53	5	6
5	19	4	4	10	21	6	7
6	56	4	5	12	12	24	28
7	37	5	5	14	14	49	56
8	53	5	6	16	19	30	42
9	16	6	6	18	37	31	43
10	21	6	7	20	16	32	44
11	15	12	14	22	15	64	20

Tabla 3.5: Algoritmo ρ de Pollard.

Como en la Tabla 3.5 tenemos que $x_{11} = x_{22}$, entonces

$$r = 14 - 22 \equiv 64 \mod 70, \qquad u = 64 - 12 = 62$$

y tenemos que resolver la ecuación

$$64x \equiv 62 \mod 70. \tag{3.7}$$

Sin embargo, como 64 y 70 no son coprimos entre sí, para poder obtener x tenemos que llegar a una ecuación equivalente. Si calculamos la Identidad de Bézout obtenemos que

$$-12 \cdot 64 + 11 \cdot 70 = 2. \tag{3.8}$$

Por tanto, si dividimos por 2 en los dos miembros de la ecuación (3.7) obtenemos la ecuación equivalente:

$$32x \equiv 26 \mod 35,$$

en la que 32 y 35 sí son coprimos entre sí. Ahora, para encontrar el inverso de 32, lo único que hay que hacer es dividir todo por 2 en la Identidad de Bézout (3.8) para llegar a que el inverso que buscamos es el número -12. Así que:

$$x \equiv 26 \cdot (-12) \equiv 3 \mod 35.$$

Entonces, $7^3 \equiv 59 \mod 71$.

3.6. Problema de Diffie-Hellman

El Protocolo de Intercambio de Claves de Diffie-Hellman visto en 2.2.10 se puede generalizar cambiando $\mathbb{Z}_p^*$ por cualquier otro semigrupo. Es decir, Alicia y Bob podrían fijar un semigrupo S de forma que el conjunto de claves se identificara con S. Para fijar una clave se pondrían de acuerdo en un elemento g de S que sería información pública y ambos elegirían de forma privada dos enteros a y b de forma que se comunicarían $A = g^a$ y $B = g^b$ de forma pública y fijarían la clave acordada como $k = g^{ab}$, calculando ambos de forma privada $A^b = k$ y $B^a = k$. Esto nos llevaría al siguiente problema:

Definición 3.6.1. Sea S un semigrupo finito y sea α un elemento de S. El llamado *Problema de Diffie-Hellman* es el siguiente: dados α^a y α^b (con a y b enteros positivos), encontrar α^{ab}.

Supongamos que pudiésemos resolver el Problema del Logaritmo Discreto de forma eficiente. Entonces, dado α^a (o α^b) en el Problema de Diffie-Hellman, podríamos calcular a (o b) resolviendo el Problema del Logaritmo Discreto. Después, para obtener α^{ab}, sería tan sencillo como calcular $(\alpha^b)^a$ (o $(\alpha^a)^b$). Esto establece una clara relación entre ambos problemas, ya que el Problema de Diffie-Hellman se puede reducir al del Logaritmo Discreto como hemos visto.

Capítulo 4

Protocolo de intercambio de claves con anillos de grupo

En este capítulo vamos examinar el nuevo protocolo de intercambio de claves propuesto por D. Kahrobaei, C. Koupparis y V. Shpilrain en el artículo *Public Key Exchange Using Matrices Over Group Rings*. Para ello, las referencias que usaré serán [3], [13], [15], [18] y [24].

4.1. Motivación

La primera pregunta que debemos hacernos antes de profundizar en el artículo es, ¿por qué es necesario buscar un nuevo protocolo de intercambio de claves? Los autores nos exponen su principal motivo en la introducción de su artículo:

- Si se realiza el Protocolo de Diffie-Hellman, hay algunas desventajas al trabajar con $\mathbb{Z}_p$. Estas desventajas se deben a que el número p y las potencias a y b deben ser números enormes para que el protocolo sea útil; y esto hace que las cuentas necesarias no sean muy eficientes, lo que puede ocasionar serios problemas si el ordenador no es lo suficientemente potente.

El nuevo protocolo que ellos proponen está basado en la generalización vista en la Sección 3.6 tomando el semigrupo de matrices sobre un anillo de grupo, más concretamente, tomando $S = \mathcal{M}_s(\mathbb{Z}_n\mathcal{S}_m)$, donde $\mathbb{Z}_n$ es el anillo de enteros módulo n y $\mathcal{S}_m$ es el grupo simétrico de orden m. Además, también en la introducción, los autores explican las siguientes ventajas de su protocolo:

- Incluso usando matrices de tamaño 2×2 o de tamaño 3×3 sobre $\mathbb{Z}_7\mathcal{S}_5$, ya se tiene un gran espacio para las claves: $(7^{5!})^4 = 7^{480} \approx 10^{406}$ elementos distintos en las matrices 2×2 y $(7^{5!})^9 = 7^{1080} \approx 10^{913}$ elementos distintos en las matrices 3×3.

- En $\mathcal{M}_s(\mathbb{Z}_7\mathcal{S}_5)$, cada entrada de la matriz puede representarse mediante una sucesión de 120 coeficientes en la que cada coeficiente ocupa 3 *bits* (puesto que si escribimos los números del 0 al 6 en binario usaremos 3 *bits* como mucho). Por tanto, cada entrada de la matriz puede codificarse como una cadena formada por $120 \cdot 3 = 360$ *bits* de información. Además, si estamos en matrices de tamaño 2×2 (resp. 3×3), cada matriz tiene cuatro entradas (resp. 9 entradas), es decir, necesitarmos $360 \cdot 4 = 1440$ *bits* (resp. $360 \cdot 9 = 3240$ *bits*) para almacenar la matriz clave. Por tanto, las claves en este protocolo tienen aproximadamente el mismo tamaño que en el Protocolo de Diffie-Hellman (guardar un entero de longitud 300 en base 10 necesita aproximadamente unos 997 *bits*). Sin embargo, si sólo almacenamos

los coeficientes distintos de cero, podremos reducir esto en una séptima parte, es decir, para guardar la clave necesitaremos aproximadamente 1230 *bits* (resp. 2780 *bits*).

- La multiplicación de matrices sobre $\mathbb{Z}_7\mathcal{S}_5$ se puede hacer de una forma muy eficiente: se pueden precalcular las multiplicaciones de los elementos de $\mathcal{S}_5$ y ponerlas en una tabla. Así, cuando multipliquemos las matrices no habrá que hacer ninguna multiplicación en $\mathcal{S}_5$ porque simplemente habrá que buscar el resultado del producto en la tabla y después multiplicar los coeficientes de $\mathbb{Z}_7$ (que es una cuenta rápida). Además, los autores también proponen usar el algoritmo de cuadrar y multiplicar explicado en el Ejemplo 2.2.16. En el artículo se afirma que esta forma de multiplicar es mucho más eficiente que la usada en el Protocolo de Diffie-Hellman en $\mathbb{Z}_p$ con p grande.

4.2. Eficacia del protocolo

Una vez explicado el nuevo protocolo, hay que ver si de verdad es útil. Para ello, entre otras cosas, hay que ver cómo es la velocidad de computación y si se satisfacen ciertas condiciones que lo hagan ser un método seguro. Después, veremos argumentos a favor de por qué los principales ataques contra protocolos no funcionan para este nuevo protocolo.

Para que el método sea verdaderamente útil necesitamos dos cosas principales:

1. Alicia y Bob deben tardar poco tiempo en calcular las potencias.
2. Si Eva quisiera encontrar la clave para poder descifrar el mensaje que se estén enviando Alicia y Bob, debería suceder que a pesar de haber conseguido (M, M^a, M^b) no pueda obtener M^{ab} de forma sencilla.

Ahora voy a realizar los experimentos explicados en el artículo ya citado de Kahrobaei, Koupparis y Shpilrain. Además, como ellos, los realizaré sobre $M_2(\mathbb{Z}_7\mathcal{S}_5)$ suponiendo que si se cumplen las propiedades para matrices de tamaño 2×2 también se cumplirán para matrices de tamaño 3×3. Sin embargo, en el caso de los experimentos sobre velocidad de computación 4.2.1, haré los experimentos para matrices de tamaño 2×2 y 3×3.

Aunque mi intención es reproducir todos los experimentos igual a como los hicieron los autores del artículo, como no puedo disponer de los códigos que usaron ellos he tenido que crearlos yo con una diferencia: los autores realizaron sus cálculos en *C++*, mientras que yo los he realizado en GAP [9]. Por otro lado mi tutor y yo hemos considerado oportuno ampliar o mejorar algunos de los experimentos del artículo.

4.2.1. Velocidad de computación

Primero vamos a ver que un ordenador tarda poco tiempo en elevar matrices aleatorias $M \in \mathcal{M}_s(\mathbb{Z}_n\mathcal{S}_m)$ a exponentes razonablemente grandes. El tiempo de cálculo es el dado por el algoritmo *Speed* (para más detalles sobre el código usado ver la Sección A.2 del anexo).

En vez de tan solo repetir el experimento tal y como está planteado en el artículo, he decidido realizarlo de tres formas distintas. Esto es debido a que no sé qué es lo que hace GAP internamente al hacer multiplicaciones y potencias, y podría pasar que su método fuera más eficaz en tiempo que el que yo he hecho a partir de lo que proponen los autores. Veamos el experimento con cada una de las formas de hacer potencias y multiplicar. Además, como ellos, cambiaré $\mathbb{Z}_n$ y el exponente al que elevo la matriz aleatoria pero siempre en el grupo $\mathcal{S}_5$.

- En la primera forma he usado la multiplicación de elementos de $M_s(\mathbb{Z}_7\mathcal{S}_5)$ y de elevar potencias de GAP, y la tabla obtenida ha sido la Tabla 4.1.

Tamaño de la matriz	$\mathbb{Z}_n$	Exponente	Tiempo medio (en segundos)
2×2	2	10^{10}	0.74
2×2	3	10^{10}	1.35
2×2	5	10^{10}	1.97
2×2	7	10^{10}	2.33
2×2	2	10^{100}	7.13
2×2	3	10^{100}	13.66
2×2	5	10^{100}	20.54
2×2	7	10^{100}	24.07
3×3	2	10^{10}	2.39
3×3	3	10^{10}	4.87
3×3	5	10^{10}	6.88
3×3	7	10^{10}	8.11
3×3	2	10^{100}	24.40
3×3	3	10^{100}	46.25
3×3	5	10^{100}	68.55
3×3	7	10^{100}	80.59

Tabla 4.1: Velocidades (primera forma).

- En la segunda forma he usado la multiplicación de elementos de $M_s(\mathbb{Z}_7\mathcal{S}_5)$ de GAP y la forma de elevar potencias dada por el método *Potencia* del anexo A.1.2. En este caso los resultados se pueden ver en la Tabla 4.2.

Tamaño de la matriz	$\mathbb{Z}_n$	Exponente	Tiempo medio (en segundos)
2×2	2	10^{10}	0.76
2×2	3	10^{10}	1.47
2×2	5	10^{10}	2.19
2×2	7	10^{10}	2.58
2×2	2	10^{100}	7.53
2×2	3	10^{100}	13.80
2×2	5	10^{100}	20.71
2×2	7	10^{100}	24.15
3×3	2	10^{10}	2.48
3×3	3	10^{10}	4.96
3×3	5	10^{10}	7.39
3×3	7	10^{10}	8.58
3×3	2	10^{100}	24.51
3×3	3	10^{100}	46.90
3×3	5	10^{100}	69.12
3×3	7	10^{100}	81.77

Tabla 4.2: Velocidades (segunda forma).

- Por último, en la tercera forma he usado la multiplicación de elementos de $M_s(\mathbb{Z}_7\mathcal{S}_5)$ del método *MultMat* y la forma de elevar potencias dada por el método *PotenciaMat* del anexo A.1.3, obteniendo la siguiente Tabla 4.3.

Tamaño de la matriz	$\mathbb{Z}_n$	Exponente	Tiempo medio (en segundos)
2×2	2	10^{10}	12.05
2×2	3	10^{10}	23.77
2×2	5	10^{10}	37.58
2×2	7	10^{10}	43.27
2×2	2	10^{100}	116.77
2×2	3	10^{100}	230.70
2×2	5	10^{100}	353.41
2×2	7	10^{100}	416.76
3×3	2	10^{10}	40.55
3×3	3	10^{10}	77.95
3×3	5	10^{10}	122.11
3×3	7	10^{10}	141.77
3×3	2	10^{100}	396.60
3×3	3	10^{100}	784.07
3×3	5	10^{100}	1189.89
3×3	7	10^{100}	1420.22

Tabla 4.3: Velocidades (tercera forma).

Ahora, veamos los datos proporcionados por los autores en su artículo:

Tamaño de la matriz	$\mathbb{Z}_n$	Exponente	Tiempo medio (en segundos)
2×2	2	10^{10}	0.06
2×2	3	10^{10}	0.06
2×2	5	10^{10}	0.06
2×2	7	10^{10}	0.06
2×2	2	10^{100}	0.58
2×2	3	10^{100}	0.58
2×2	5	10^{100}	0.58
2×2	7	10^{100}	0.59
3×3	2	10^{10}	0.19
3×3	3	10^{10}	0.20
3×3	5	10^{10}	0.20
3×3	7	10^{10}	0.20
3×3	2	10^{100}	1.95
3×3	3	10^{100}	1.95
3×3	5	10^{100}	1.94
3×3	7	10^{100}	1.94

Tabla 4.4: Velocidades dadas por el artículo.

Por un lado, si nos fijamos en la Tabla 4.3 y en la Tabla 4.4, está claro que, a pesar de usar las mismas ideas, los métodos empleados por D. Kahrobael, C. Koupparis y V. Shpilrain son mucho más rápidos que los que he programado. Esto es debido a que ellos han usado el lenguaje *C++* que es mucho más rápido que GAP porque es un lenguaje compilado. Por otro lado, si comparamos las cuatro tablas entre sí, la más rápida es la de los autores (Tabla 4.4), y la segunda más rápida es la dada por la primera forma (Tabla 4.1), es decir, cuando GAP usa sus propios métodos de multiplicación y exponenciación. Debido a esto y a que no puedo disponer de los códigos usados por los autores, a partir de ahora realizaré todos los cálculos con la primera forma.

Ahora, fijada la forma de hacer los cálculos nos surge una pregunta: ¿de qué forma aumenta el tiempo que tarda GAP en elevar una matriz a un exponente en un anillo de grupo como los que estamos viendo? Vamos a verlo con una gráfica en la que vamos a ir cambiando el anillo de coeficientes de forma que los cálculos se vayan haciendo en $\mathbb{Z}_n\mathcal{S}_5$ para ciertos enteros n.

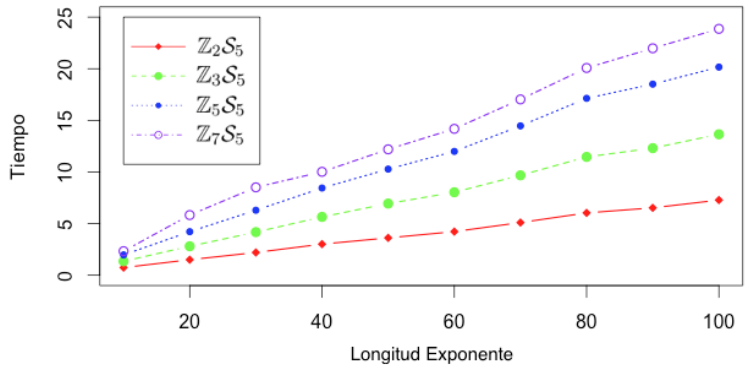


Figura 4.1: Gráfica de velocidad de computación, donde cada punto (x, y) representa que se tardan y segundos de media en elevar una matriz aleatoria al exponente 10^x en los anillos de grupo indicados en la leyenda.

Mirando la gráfica de la Figura 4.1 parece claro que, fijado un anillo de grupo, el tiempo que se tarda en elevar una matriz aleatoria aumenta con respecto a la longitud el exponente.

4.2.2. Hipótesis computacional y de decisión de Diffie-Hellman

Una vez vista la velocidad de computación, supongamos que Eva está tratando de obtener M^{ab} conociendo (M, M^a, M^b). ¿Lo conseguirá? Para saberlo necesitaremos varias definiciones.

Definición 4.2.1. Se dice que F es un *algoritmo de Diffie-Hellman de* G si, dados como entradas un semigrupo G y tres elementos $g, x, y \in G$, proporciona como salida otro elemento de G que denotaremos $F(g, G, x, y)$ de forma que si $g \in G$ y a y b son enteros positivos entonces $F(g, G, x, y) = g^{ab}$.

Diremos que un semigrupo G *satisface la Hipótesis Computacional de Diffie-Hellman* (la *hipótesis CDH*) si no existe ningún algoritmo de Diffie-Hellman probabilístico eficiente, es decir, de tiempo polinomial. Vamos a ver esto de forma más rigurosa.

Definición 4.2.2. Supongamos que tenemos un algoritmo probabilístico F de tiempo polinomial tal que dados como entradas un semigrupo G y tres elementos $g, x, y \in G$ proporciona como salida un elemento de G denotado por $F(g, G, x, y)$. Diremos que F es un *algoritmo CDH*

para G si satisface, para un cierto $\alpha > 0$ fijo y para un $n \in \mathbb{N}$ lo suficientemente grande, la siguiente condición:

$$\mathcal{P}[F(g, G, g^a, g^b) = g^{ab}] > \frac{1}{n^\alpha},$$

donde la probabilidad está sobre la elección uniformemente aleatoria de a y b (con $0 \leq a, b \leq n$).

Obviamente, Alicia y Bob quieren que no exista tal algoritmo para que su clave sea lo suficientemente secreta y segura. Por tanto, diremos que un semigrupo G *satisface la hipótesis CDH* si no existe ningún algoritmo CDH para G.

Sin embargo, aunque tengamos un semigrupo que satisfaga la hipótesis CDH, esto no nos garantizará que el protocolo sea útil para la criptografía, ya que puede suceder que aunque Eva no sea capaz de recuperar toda la información sí pueda recuperar parte de los bits de g^{ab}.

Entonces queremos que tampoco exista ningún algoritmo que sea capaz de conocer muchos bits de información de la clave, por tanto nos interesará que exista una especie de cota sobre la cantidad de información que puede averiguar Eva. Esto da lugar a la siguiente definición, la definición de la *Hipótesis de Decisión de Diffie-Hellman* (la *hipótesis DDH*).

Definición 4.2.3. Sea F' un algoritmo probabilístico en tiempo polinomial diseñado para resolver el siguiente problema de decisión: decidir si dados un semigrupo G, $g \in G$ y $x, y, z \in G$ tales que $x = g^a, y = g^b$ y $z = g^c$ (con $a, b, c \in \mathbb{N}$) se cumple $g^{ab} = g^c$. Diremos que F' es un *algoritmo DDH para* G si satisface, para un cierto $\alpha > 0$ fijo y para un $n \in \mathbb{N}$ lo suficientemente grande, la siguiente condición:

$$\left|\mathcal{P}[F'(g, G, g^a, g^b, g^{ab}) = \text{"}Verdadero\text{"}] - \mathcal{P}[F'(g, G, g^a, g^b, g^c) = \text{"}Verdadero\text{"}]\right| > \frac{1}{n^\alpha},$$

donde la probabilidad está sobre la elección uniformemente aleatoria de a, b y c (con $0 \leq a, b, c \leq n$).

De nuevo, Alicia y Bob querrán que no exista tal algoritmo, así que diremos que un semigrupo G *satisface la hipótesis DDH* si no existe ningún algoritmo DDH para G. Básicamente, decir que un grupo satisface la hipótesis DDH significa que las distribuciones de (g^a, g^b, g^{ab}) y de (g^a, g^b, g^c) son indistinguibles (eligiendo a, b y c de forma arbitraria), o lo que es lo mismo, que no existe ningún algoritmo tal que dados g, g^a, g^b y g^c sea capaz de decir si $g^{ab} = g^c$ con suficiente seguridad.

Una vez vistas las definiciones, vamos a ver si el nuevo protocolo las satisface o no. Para comprobar si se cumple la Hipótesis de Decisión de Diffie-Hellman queremos ver que la distribución que sigue la tripla (M^a, M^b, M^{ab}) es completamente indistinguible de la distribución que sigue (M^a, M^b, M^c) con M, a, b y c aleatorios. Para ello, he realizado tres experimentos a partir de los comentarios que han hecho los autores en el artículo sobre los experimentos que realizaron ellos.

En el primer experimento comprobaremos si la distribución que sigue una matriz de la forma M^{ab} es la misma que la que sigue la matriz M^c y en el segundo veremos lo análogo con la matriz M^a y con una matriz aleatoria cualquiera N. Con estos dos experimentos sabremos que cada componente por separado de la tripla (M^a, M^b, M^{ab}) es completamente indistinguible de una matriz aleatoria cualquiera. Por último, en el tercer experimento veremos si la tripla (M^a, M^b, M^{ab}) sigue una distribución que es similar a la de una tripla de la forma (N_1, N_2, N_3) con N_1, N_2 y N_3 matrices aleatorias independientes y por tanto, si su distribución es indistinguible de la distribución que tiene la tripla (M^a, M^b, M^c). Como ya he comentado, estas comprobaciones las haremos sobre $\mathcal{M}_2(\mathbb{Z}_7\mathcal{S}_5)$.

Notación 4.2.4. Como por lo general usaremos matrices de tamaño 2×2, usaremos la notación usual de las matrices, es decir:

$$M = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

Experimento 1

Este experimento va a tener dos partes:

- En la primera parte escogeremos una cierta cantidad de matrices aleatorias y las elevaremos a $a \cdot b$, siendo a y b dos enteros pertenecientes a un intervalo predeterminado y elegidos de forma aleatoria y uniforme para cada matriz. Una vez hecha esa operación, miraremos por ejemplo la primera entrada de esas matrices resultantes y anotaremos cuántas veces aparece en total cada una de las permutaciones de $\mathcal{S}_5$. Esto lo haremos con las cuatro entradas de las matrices.
- En la segunda parte escogeremos la misma cantidad de matrices aleatorias que antes y las elevaremos a un entero c que de nuevo pertenecerá a un intervalo predeterminado y se elegirá de forma aleatoria y uniforme para cada matriz. Después, contaremos las permutaciones de la misma manera que hicimos en la primera parte de este experimento.

Vamos a ver un ejemplo para comprender la forma de "contar permutaciones" mejor. El ejemplo lo voy a hacer con la segunda parte del experimento; con la primera parte es análogo.

Ejemplo 4.2.5. Supongamos que, después de haber seleccionado dos matrices aleatorias M_1 y M_2 y unos enteros c_1 y c_2, hacemos los cálculos $M_1^{c_1}$ y $M_2^{c_2}$ y obtenemos las siguientes matrices N_1 y N_2:

$$N_1 = \begin{pmatrix} 3 \cdot (4\ 5) + 2 \cdot (3\ 5) & 4 \cdot (1\ 5\ 4) \\ (1\ 4) + 5 \cdot (2\ 5) & 2 \cdot (1\ 3\ 5) + (4\ 5) \end{pmatrix}$$

$$N_2 = \begin{pmatrix} 6 \cdot (4\ 5) + 3 \cdot (1\ 4) & 2 \cdot (1\ 4) \\ 5 \cdot (2\ 5) + (1\ 5)(2\ 4) & (2\ 4) + 5 \cdot (2\ 4\ 3) \end{pmatrix}$$

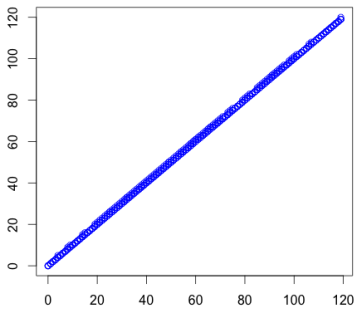
Supongamos que estamos mirando la entrada a_{11} de las matrices. Entonces, diríamos que la permutación $(4\ 5)$ está dos veces, las permutaciones $(3\ 5)$ y $(1\ 4)$ están una vez y el resto de permutaciones de $\mathcal{S}_5$ están cero veces. En el experimento haremos esto con quinientas matrices en vez de con dos y lo haremos con cada una de las entradas de las matrices.

Antes de exponer los resultados obtenidos, es necesario hacer una aclaración: en este primer experimento, los autores toman a y b de forma aleatoria uniforme en el intervalo $[10^{22}, 10^{28}]$ y c de forma aleatoria uniforme en el intervalo $[10^{44}, 10^{55}]$ para que así c tenga aproximadamente el mismo tamaño que el producto ab. Sin embargo, por limitaciones del software GAP tengo que poner exponentes más pequeños, así que el intervalo para a y b será $[10^6, 10^8]$, y el intervalo para c será $[10^{12}, 10^{16}]$. Los detalles sobre el código que he usado están en el anexo A.3.1.

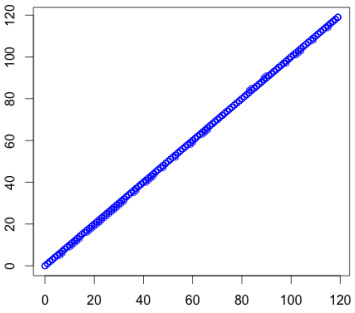
Una vez obtenidos los datos resultantes de las dos partes del experimento vamos a ver, entrada por entrada, si las distribuciones que siguen las matrices de la forma M^{ab} son similares a las distribuciones que siguen las matrices M^c. Esta parte la voy a ver usando los conocimientos adquiridos a través de asignaturas de probabilidad y estadística (para más información consultar el libro *Elementos de Probabilidades* [29]).

Para ver la igualdad de las distribuciones vamos a crear una gráfica Q-Q. Es decir, primero crearemos las funciones de distribución acumuladas, F y G, de los resultados obtenidos en la primera y segunda parte, respectivamente, del experimento. Después, con esas funciones

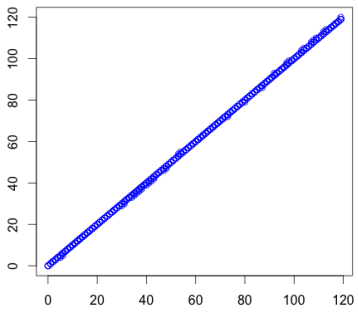
dibujaremos una gráfica Q-Q en la que el eje x representará $F^{-1}(p)$ y el eje y representará $G^{-1}(p)$ con $p \in [0,1]$. Al hacer el primer experimento (con matrices de tamaño 2×2 y entradas en $\mathbb{Z}_7\mathcal{S}_5$) y tomando quinientos puntos $p \in [0,1]$, las gráficas Q-Q obtenidas han sido las siguientes:



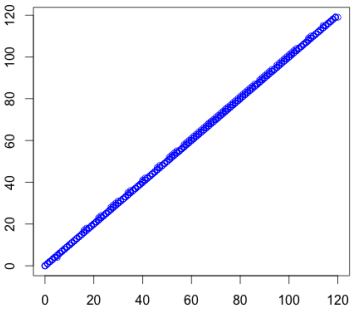
(a) Entrada a_{11}.



(b) Entrada a_{12}.



(c) Entrada a_{21}.



(d) Entrada a_{22}.

Figura 4.2: Q-Q gráficas.

Observando las gráficas de la Figura 4.2 podemos ver que nos salen líneas rectas, por tanto parece claro que las distribuciones son iguales, es decir, no existe ninguna diferencia entre las matrices de la forma M^{ab} y las matrices de la forma M^c.

Experimento 2

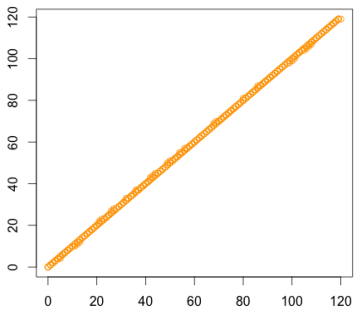
De nuevo, este experimento va a tener dos partes:

- En la primera parte escogeremos matrices aleatorias y las elevaremos a un entero a perteneciente a un intervalo predeterminado y elegido de forma aleatoria y uniforme. Después, contaremos las permutaciones de cada entrada como hicimos en el experimento 1.
- En la segunda parte escogeremos matrices aleatorias y contaremos las permutaciones de cada entrada como en la primera parte.

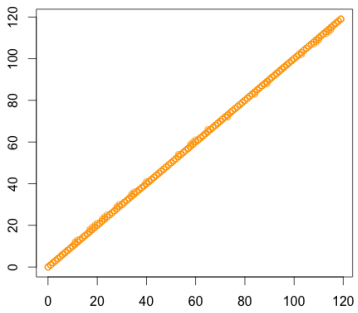
Después, para poder ver si las distribuciones son similares realizaré las gráficas Q-Q explicadas en el experimento 1 para cada una de las entradas.

En este experimento también sucede que, por limitaciones del software GAP, no puedo tomar el intervalo dado por los autores $[10^{44}, 10^{55}]$ para a y tengo que tomar un intervalo con los rangos más pequeños. En este caso tomaré $a \in [10^{12}, 10^{16}]$.

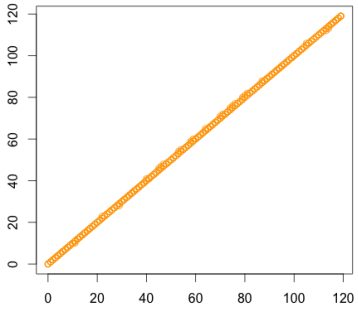
Las gráficas obtenidas, de nuevo con quinientos puntos $p \in [0, 1]$ son:



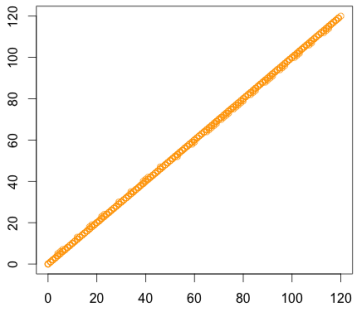
(a) Entrada a_{11}.



(b) Entrada a_{12}.



(c) Entrada a_{21}.



(d) Entrada a_{22}.

Figura 4.3: Q-Q gráficas.

Si miramos las gráficas de la Figura 4.3 podemos ver líneas rectas, por tanto las distribución que siguen las matrices de la forma M^a es la misma que la que siguen las matrices aleatorias N. En otras palabras, Eva no tiene ninguna información sobre el valor de a al ver la matriz M^a.

Experimento 3

En este experimento estudiaremos si la distribución de triplas de la forma (M^a, M^b, M^{ab}), con a, b enteros aleatorios escogidos de forma uniforme en un intervalo y M una matriz aleatoria de $\mathcal{M}_2(\mathbb{Z}_7\mathcal{S}_5)$, es similar a la distribución de triplas de la forma (M_1, M_2, M_3) con M_1, M_2 y M_3 matrices aleatorias de $\mathcal{M}_2(\mathbb{Z}_7\mathcal{S}_5)$.

Para estudiar estas distribuciones lo haremos por medio de unas tablas de tamaño $7^3 \times 120$ que serán las que nos devuelvan las funciones del anexo A.3.3. En estas tablas, las filas representan todas las posibles combinaciones de tres elementos de $\mathbb{Z}_7$ y las columnas representan las ciento veinte permutaciones de $\mathcal{S}_5$. El número que haya en la posición i, j indicará cuántas veces se ha dado, en la tripla de matrices, que la permutación j haya tenido como coeficientes la

combinación de tres coeficientes dada por i. Como es un poco complicado entender qué indican exactamente las tablas, vamos a ver un ejemplo sencillo.

Ejemplo 4.2.6. Para entender el comportamiento de estas tablas, vamos a ver un ejemplo de cómo la rellenaríamos. Supongamos que estamos trabajando con matrices de $\mathcal{M}_2(\mathbb{Z}_2\mathcal{S}_3)$ y que nos interesa la entrada a_{11} de las matrices.

Antes de nada necesitaremos dos listas. La primera de ellas estará formada por todas las posibles combinaciones de tres elementos de $\mathbb{Z}_2$, y la segunda estará formada por todas las permutaciones de $\mathcal{S}_3$. Es decir, la primera lista será

$$\{000, 001, 010, 011, 100, 101, 110, 111\},$$

mientras que la segunda será

$$\{(\ ), (2\ 3), (1\ 2), (1\ 2\ 3), (1\ 3\ 2), (1\ 3)\}.$$

Ahora, supongamos que al mirar las entradas $1, 1$ de las matrices de la primera tripla que calculamos obtenemos lo siguiente:

$$\Big((2\ 3) + (1\ 3)\,,\ (2\ 3) + (1\ 2) + (1\ 3\ 2)\,,\ (1\ 2\ 3) \Big)$$

La siguiente tabla muestra qué combinaciones de coeficientes asignará el algoritmo a cada permutación.

Permutación	Coeficientes
( )	(0, 0, 0)
(2 3)	(1, 1, 0)
(1 2)	(0, 1, 0)
(1 2 3)	(0, 0, 1)
(1 3 2)	(0, 1, 0)
(1 3)	(1, 0, 0)

Tabla 4.5: Ejemplo de asignación de coeficientes.

Entonces, tendríamos que rellenar la tabla de la siguiente forma (antes de empezar a rellenar la tabla todas las posiciones tenían asignado el valor cero porque hemos dicho que esta era la primera tripla que obteníamos):

	( )	(2 3)	(1 2)	(1 2 3)	(1 3 2)	(1 3)
(0,0,0)	1	0	0	0	0	0
(0,0,1)	0	0	0	1	0	0
(0,1,0)	0	0	1	0	1	0
(0,1,1)	0	0	0	0	0	0
(1,0,0)	0	0	0	0	0	1
(1,0,1)	0	0	0	0	0	0
(1,1,0)	0	1	0	0	0	0
(1,1,1)	0	0	0	0	0	0

Tabla 4.6: Ejemplo de experimento 3 (con la primera tripla).

Si observamos la Tabla 4.6 por columnas, dada una permutación j hemos sumado 1 en el coeficiente i-ésimo que le hemos asignado en la Tabla 4.5 (es decir, en la entrada i, j sumamos 1) y en el resto de coeficientes hemos dejado lo que ya había (es decir, en el resto de entradas i', j con $i' \neq i$ dejamos lo que ponga). Recordemos que la tabla estaba inicialmente con todas las entradas iguales a cero.

De nuevo, en este experimento también tendré que, por limitaciones de GAP, tomar un intervalo distinto para los valores a y b. Además, en este caso el intervalo lo he tenido que tomar muy pequeño (he tomado el intervalo $[10, 20]$) debido al tiempo que tardaba el ordenador en hacer los cálculos, ya que, para hacerlo con doce mil cinco triplas de matrices tardó tres horas. Los detalles del código usado están en el anexo A.3.3.

Antes de ver cómo son las tablas obtenidas, vamos a pensar qué valores deberían salir en ellas. Si suponemos que los coeficientes de las triplas están distribuidos de forma uniforme sobre $\mathbb{Z}_7^3$, cada combinación sucederá con probabilidad $1/7^3$. Como hemos hecho el experimento con doce mil cinco triplas de matrices, cada combinación de coeficientes debería salir aproximadamente $12{,}005/7^3 = 35$ veces.

Una vez visto el valor aproximado que debería salir en cada entrada de las tablas, debo hacer una observación: debido a las dimensiones de las tablas obtenidas (343×120), en el trabajo sólo voy a escribir una sección de una de las dos tablas, en concreto de la tabla que resulta de hacer el experimento para triplas de la forma (M^a, M^b, M^{ab}). Por otro lado, usaré la notación σ_i para referirme a las permutaciones y e_i para referirme a las combinaciones de coeficientes.

	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9	σ_{10}	σ_{11}	σ_{12}	σ_{13}	σ_{14}	σ_{15}	σ_{16}	σ_{17}
e_1	66	52	59	70	59	46	54	49	52	53	63	57	52	47	43	43	63
e_2	54	52	46	68	59	58	54	56	51	50	67	42	50	62	49	50	57
e_3	55	51	56	37	42	64	54	51	60	58	55	57	53	50	53	50	49
e_4	57	59	52	53	39	46	50	60	51	50	56	59	50	52	58	53	62
e_5	64	52	46	50	52	54	55	47	61	44	46	47	56	52	61	65	45
e_6	60	63	69	64	53	59	48	61	59	72	60	57	61	52	48	61	37
e_7	60	50	46	40	54	47	59	60	53	57	60	51	45	59	66	38	39
e_8	28	28	35	37	28	37	32	32	29	33	35	25	27	36	31	28	38
e_9	32	23	31	37	24	28	47	36	31	37	37	31	28	38	46	34	34
e_{10}	30	42	32	26	28	40	33	30	38	25	35	36	30	28	30	30	30
e_{11}	40	32	42	31	35	34	36	35	29	25	27	30	24	35	21	38	37
e_{12}	36	31	33	28	24	42	29	44	29	24	34	35	28	28	32	32	24
e_{13}	38	19	27	32	33	20	33	33	29	33	38	38	30	29	23	30	29
e_{14}	33	33	24	27	35	31	35	43	38	38	38	31	33	29	35	29	34
e_{15}	40	35	29	25	31	27	30	28	28	39	30	27	32	57	28	36	33
e_{16}	32	35	36	33	32	31	39	22	46	24	25	38	26	35	27	41	27
e_{17}	30	25	35	35	23	39	27	37	39	38	29	35	35	31	28	31	31
e_{18}	19	19	36	28	27	28	33	35	31	45	26	40	32	24	44	34	31
e_{19}	29	36	30	36	34	30	39	26	35	32	25	46	34	39	30	28	36
e_{20}	37	28	30	35	33	27	35	33	41	45	31	30	34	32	34	33	38
e_{21}	36	37	33	29	34	30	25	29	27	28	25	27	30	32	23	28	29
e_{22}	46	22	34	34	33	32	33	35	40	22	33	35	39	28	33	34	30
e_{23}	26	37	29	34	32	30	30	35	36	32	30	36	24	29	30	42	29

Tabla 4.7: Distribución de los coeficientes de las triplas (continúa en la página siguiente).

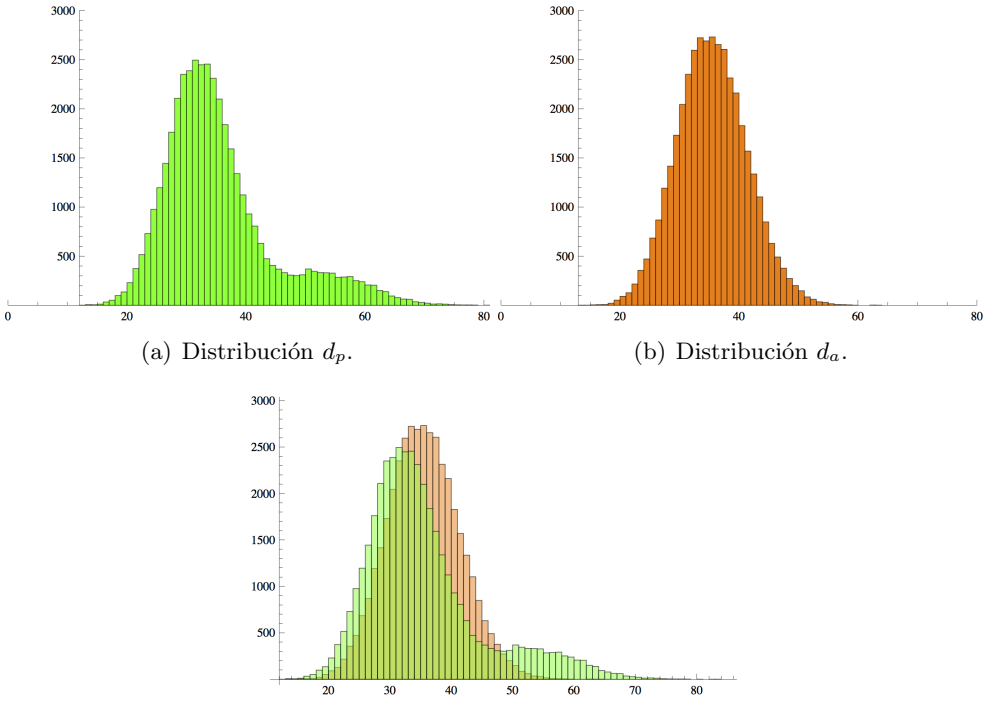
e_{24}	29	21	26	36	36	36	30	25	46	42	26	42	30	30	25	30	33
e_{25}	27	31	41	34	27	42	38	31	37	39	23	35	23	31	31	28	37
e_{26}	32	22	32	28	33	36	39	36	26	30	30	42	28	20	36	36	30
e_{27}	22	41	34	36	38	37	30	29	22	40	40	39	38	22	33	32	36
e_{28}	33	43	33	38	28	38	29	30	25	27	42	37	31	22	26	27	30
e_{29}	41	32	24	39	31	29	23	41	26	32	29	30	44	32	25	18	30
e_{30}	40	29	36	41	31	28	34	37	27	25	29	30	36	27	33	40	37
e_{31}	34	33	27	38	30	35	40	32	33	39	33	31	13	37	28	38	35
e_{32}	22	43	36	27	35	29	36	24	30	28	35	29	34	29	26	32	35
e_{33}	29	34	26	37	32	40	37	36	27	27	21	46	33	36	40	29	25
e_{34}	33	25	33	23	36	27	39	32	30	30	23	44	33	32	25	26	36
e_{35}	32	24	33	34	37	35	36	28	43	24	40	32	22	34	31	46	35
e_{36}	35	29	33	33	31	37	23	29	38	27	41	35	26	31	24	32	33
e_{37}	37	36	37	30	38	28	31	26	34	25	43	29	36	41	28	25	23
e_{38}	35	29	27	35	24	33	24	33	42	18	41	33	32	27	34	32	26
e_{39}	32	29	31	30	20	40	33	38	32	35	45	42	33	22	32	35	31
e_{40}	31	31	19	31	28	35	20	27	26	35	36	24	27	30	24	28	39
e_{41}	42	38	35	31	29	34	22	32	34	25	23	27	34	28	34	24	31
e_{42}	25	28	24	28	32	41	31	29	39	32	40	31	25	26	40	39	30
e_{43}	36	27	26	35	33	23	43	24	30	38	34	38	38	31	29	35	34
e_{44}	22	46	38	33	30	37	28	35	41	27	33	25	30	28	30	30	37
e_{45}	28	33	26	34	39	31	29	40	30	28	30	27	28	52	39	27	37
e_{46}	30	28	29	38	29	26	27	36	33	35	33	32	37	36	28	35	36
e_{47}	32	31	31	40	35	29	31	34	34	33	22	42	26	27	36	35	37
e_{48}	33	35	29	36	32	38	31	34	28	26	25	27	29	27	32	30	37
e_{49}	29	33	27	34	35	31	31	33	37	31	28	25	26	32	29	32	21
e_{50}	25	32	33	30	28	34	32	29	31	34	31	31	25	30	30	44	30
e_{51}	25	37	41	30	31	31	35	21	39	34	37	31	38	27	41	35	27
e_{52}	39	36	36	32	30	39	43	27	32	34	27	30	27	28	29	18	27
e_{53}	26	37	29	28	28	28	46	29	23	30	33	31	33	28	33	32	32
e_{54}	32	25	39	42	38	37	34	42	31	37	35	28	32	28	34	31	33
e_{55}	31	33	42	29	43	23	33	36	36	35	34	30	31	37	29	32	35

Tabla 4.8: Distribución de los coeficientes de las triplas (continuación).

Si calculamos la media de todas las entradas de la Tabla 4.7 completa y la media de las entradas de la tabla que resulta de hacer el experimento con las triplas de la forma (M_1, M_2, M_3) con un programa estadístico como *R - Studio* [22], obtenemos que la media en ambos casos es 35, así que los valores cumplen la condición de la media. Tras esto, los autores del artículo dicen que la distribución que se observa en los coeficientes de las triplas (M^a, M^b, M^{ab}) es completamente indistinguible de la distribución de los coeficientes de las triplas (M_1, M_2, M_3).

Sin embargo, a nosotros nos parece que este argumento de la media es algo débil, pues no se compara la distribución d_p de los números en la tabla obtenida cuando se usan las ternas (M^a, M^b, M^{ab}) con la correspondiente distribución d_a en el caso en el que se usan las ternas de matrices aleatorias. La p en d_p se refiere a la palabra "por" y la a en d_a se refiere a "aleatorio". Para empezar, observemos que aunque d_p y d_a tienen la misma media, sus varianzas son considerablemente diferentes. En concreto, d_p tiene varianza aproximadamente 90; mientras

que la varianza de d_a es aproximadamente 35. Esta diferencia se observa más claramente en sus histogramas, que están representados en la Figura 4.4. Además, observamos que, mientras el histograma de d_a es simétrico, el otro no lo es y hay una diferencia sustancial en los valores de la cola de d_p.



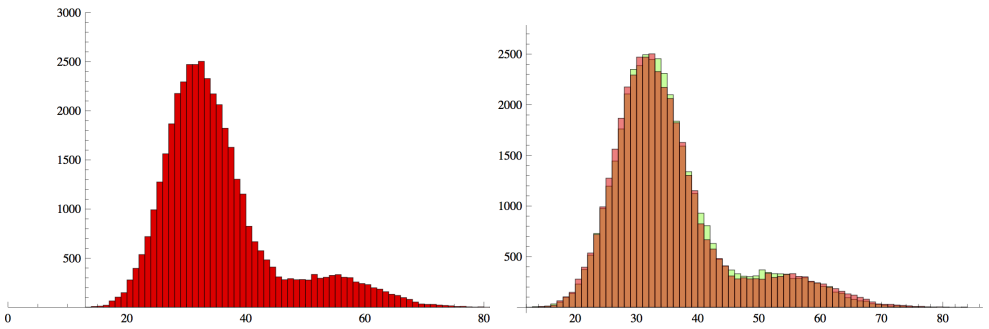
(a) Distribución d_p.

(b) Distribución d_a.

(c) Distribuciones d_p y d_a enfrentadas.

Figura 4.4: Histogramas de las distribuciones d_p y d_a.

Por otro lado, hemos realizado un experimento similar en el que hemos construido una nueva tabla, cuya distribución denotaremos por d_m (en este caso, la m viene de la palabra "más"), usando en este caso ternas del tipo (M^a, M^b, M^{a+b}). La media de d_m es de nuevo 35 y la varianza es aproximadamente 97. El histograma resultante de la distribución de d_m junto con su comparación con la distribución de d_p está representado en la Figura 4.5.



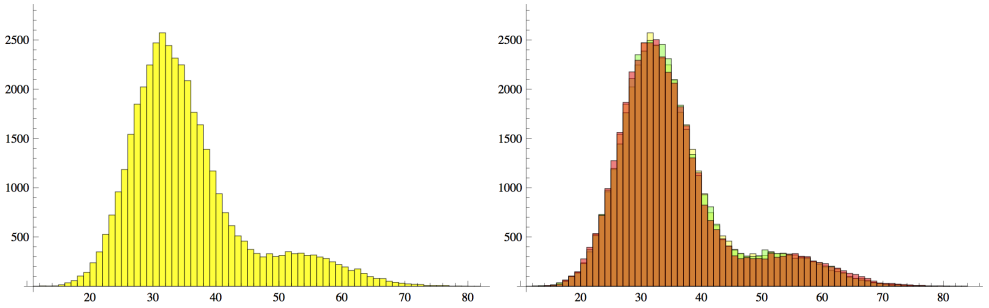
(a) Distribución d_m.

(b) Distribuciones d_m y d_p enfrentadas.

Figura 4.5: Histogramas de las distribuciones d_m y d_p.

Observamos en este caso que las distribuciones de d_p y d_m son prácticamente iguales. Esto indica que si en este experimento pudieramos obtener alguna conclusión sería que la corre-

lación entre las entradas de las triplas (M^a, M^b, M^{ab}) es tan grande como la de las triplas (M^a, M^b, M^{a+b}), pero en estas segundas, la tercera entrada se puede obtener fácilmente a partir de las otras dos ya que es el producto entre ambas. Más aún, si repetimos el experimento con triplas de la forma (M^a, M^b, M^c), los histogramas obtenidos (que se pueden observar en la Figura 4.6) muestran que las distribuciones de d_m y d_p tiene un histograma casi idéntico al de esta última distribución (a la que denotaremos d_c). Concluimos por tanto que el experimento 3 realizado por los autores no parece aportar la información que ellos dicen.



(a) Distribución d_c.

(b) Distribuciones d_c, d_p y d_m enfrentadas.

Figura 4.6: Histogramas de las distribuciones d_c, d_p y d_m.

4.2.3. Existencia de ciclos

Supongamos que Alicia escoge un entero a muy grande y hace M^a con una cierta matriz $M \in \mathcal{M}_s(\mathbb{Z}_n\mathcal{S}_m)$ y que Bob hace M^b con un entero b muy grande. Supongamos ahora que Eva encuentra dos enteros $c << a$ y $d << b$ tales que $M^c = M^a$ y $M^d = M^b$, esto no sería nada bueno, ya que entonces Eva podría calcular la clave secreta de Alicia y Bob de la siguiente forma:

$$M^{ab} = (M^a)^b = (M^c)^b = (M^b)^c = (M^d)^c = M^{cd}.$$

Queremos por tanto ver que sea poco probable que existan tales c y d pequeños. Los autores del artículo dan un argumento en el que llevan a cabo unos experimentos que, al no darles ningún resultado en varias semanas, les hacen concluir que una cota inferior para c (y para d) es 10^{10}. Según mi tutor, el procedimiento seguido por los autores se podría mejorar utilizando que el anillo usado es semisimple. En consecuencia, en vez de seguir los experimentos de los autores voy a usar los conocimientos adquiridos en las asignaturas de *Álgebra Conmutativa* y *Álgebra no Conmutativa* para ver a qué conclusión llego. Antes de nada, voy a reformular el problema: se va a tratar de, dada una matriz $M \in \mathcal{M}_2(\mathbb{Z}_7\mathcal{S}_5)$, encontrar los enteros $m \geq 0$ y $h > 0$ mínimos tales que $M^{m+1} = M^{m+h+1}$. Si recordamos el dibujo que resultaba del *método ρ de Pollard* (la Figura 3.1), en nuestro caso h será la longitud del ciclo y $m+1$ será el exponente del primer elemento a partir del cual se forma el ciclo.

Vamos a resolver el problema en un caso más general, supongamos que tenemos un semigrupo S y un elemento $s \in S$. Decimos que s *se repite* si el conjunto de las potencias de s es finito. Eso significa que existen enteros $m \geq 0$ y $h > 0$ tales que $M^{m+1} = M^{m+h+1}$. La forma más sencilla (aparentemente) para ver si hay un ciclo sería calcular $s, s^2, s^3, \ldots$ y comparar cada nuevo cálculo con todos los anteriores. Obviamente, este método seguramente no funcionaría porque tardaríamos demasiado tiempo y necesitaríamos demasiada memoria para guardar todos los elementos. Vamos a estudiar el problema para tratar de encontrar un nuevo método.

Problema 4.2.7. Queremos buscar enteros $m \geq 0$ y $h > 0$ tales que $s^{m+1} = s^{m+h+1}$, siendo m y h los menores enteros que cumplen dicha propiedad.

Obsérvese que si $s^{m+1} = s^{m+h+1}$ y n y k son enteros con $n \geq m+1$, entonces $s^n = s^{n+hk}$. Esto implica que si m es el menor entero no negativo para el que existe $h > 0$ con $s^{m+1} = s^{m+h+1}$ y h es el menor entero positivo que cumple esta propiedad, entonces $s^1, \ldots, s^m$ son distintos y para n_1, $n_2 > m$ se tiene que $s^{n_1} = s^{n_2}$ si y sólo si $n_1 \equiv n_2 \mod h$. En tal caso, la *cola de* s es por definición $\{s^1, \ldots, s^m\}$ y m es la *longitud de la cola*. El *ciclo de* s es el conjunto $\{s^{m+1}, \ldots, s^{m+h}\}$ y h es la *longitud del ciclo*. Las longitudes m y h de la cola y el ciclo de s respectivamente, están caracterizados por la siguiente propiedad:

$$s^{n_1} = s^{n_2} \Leftrightarrow n_1, n_2 > m \text{ y } n_1 \equiv n_2 \mod h$$

para cualesquiera enteros positivos distintos n_1 y n_2. Si S tiene unidad y s es invertible en S, entonces la longitud de la cola de s es 0 y la longitud del ciclo es el orden de s en S.

Lema 4.2.8. *Sean $S_1, \ldots, S_k$ semigrupos y sea $s = (s_1, \ldots, s_k) \in S_1 \times \ldots \times S_k$. Entonces s se repite si y sólo si $s_1, \ldots, s_k$ se repiten. En tal caso, la longitud de la cola de s es el máximo de las longitudes de las colas de los s_i y la longitud del ciclo de s es el mínimo común múltiplo de las longitudes de los ciclos de los s_i.*

Demostración. Se puede demostrar por inducción, por lo que basta demostrar el resultado para $k = 2$. Sean $m = \max(m_1, m_2)$ y $h = \mathrm{mcm}(\mathrm{h}_1, \mathrm{h}_2)$. Tenemos que demostrar que m y h son los mínimos enteros positivos que cumplen $s^{m+1} = s^{m+h+1}$. En primer lugar, como $m \geq m_i$ y $h_i \mid h$ tenemos $s_i^{m+1} = s_i^{m+h+1}$ para $i = 1, 2$ y por tanto $s^{m+1} = s^{m+h+1}$. Recíprocamente, sean a y b dos enteros positivos que cumplen $s^{a+1} = s^{a+b+1}$. Entonces $s_i^{a+1} = s_i^{a+b+1}$ y por tanto $a \geq m_i$ y $h_i \mid b$ para $i = 1, 2$, luego $a \geq m$ y $h \mid b$. □

Como vimos en la Observación 2.1.36, podemos descomponer $\mathbb{Z}_7\mathcal{S}_5$ como producto de matrices. Si usamos por GAP para calcular la Descomposición de Wedderburn obtenemos lo siguiente:

$$\mathbb{Z}_7\mathcal{S}_5 \simeq \mathbb{Z}_7^2 \times \mathcal{M}_4(\mathbb{Z}_7)^2 \times \mathcal{M}_5(\mathbb{Z}_7)^2 \times \mathcal{M}_6(\mathbb{Z}_7)$$

Si ahora usamos esto en nuestro caso de $\mathcal{M}_2(\mathbb{Z}_7\mathcal{S}_5)$, obtenemos:

$$\begin{aligned} \mathcal{M}_2(\mathbb{Z}_7\mathcal{S}_5) &\simeq \mathcal{M}_2\left(\mathbb{Z}_7^2 \times \mathcal{M}_4(\mathbb{Z}_7)^2 \times \mathcal{M}_5(\mathbb{Z}_7)^2 \times \mathcal{M}_6(\mathbb{Z}_7)\right) \\ &\simeq \mathcal{M}_2(\mathbb{Z}_7)^2 \times \mathcal{M}_8(\mathbb{Z}_7)^2 \times \mathcal{M}_{10}(\mathbb{Z}_7)^2 \times \mathcal{M}_{12}(\mathbb{Z}_7) \end{aligned} \tag{4.1}$$

Entonces, usando la descomposición dada por (4.1) podemos simplificar nuestro Problema 4.2.7, ya que podemos calcular m_i y h_i (con $i = 1, \ldots, 7$) para cada uno de los elementos del producto en el que se puede descomponer $\mathcal{M}_2(\mathbb{Z}_7\mathcal{S}_5)$ y después relacionarlos con el Lema 4.2.8.

Ahora, aunque hemos reducido nuestro problema a encontrar unos ciertos enteros m_i y h_i que serán más pequeños y más fáciles de encontrar que m y h, aún nos falta hacer un algoritmo que encuentre los enteros que buscamos y que sea eficiente en tiempo y memoria. Voy a usar un algoritmo que consistirá en, dado s, ir escogiendo distintos enteros x e y, con x siempre un número potencia de dos, de la manera siguiente: si por ejemplo $x = 4$, entonces y podrá tomar los valores $y = 5, 6, 7, 8$ y nos iremos preguntando si $s^x = s^{x+y}$. En el caso de haber llegado a $y = 8$(es decir, $y = 2x$), si $s^4 \neq s^{4+8}$ entonces el valor de x cambiaría a $x = 8$ y los posibles valores para y cambiarían y variarían desde 8 hasta 16.

Una vez finalizado el proceso con unos ciertos x e y, como $s^x = s^{x+y}$, podemos tomar $x' = x - 1$ y así tendremos que se cumple la igualdad del Problema 4.2.7, es decir, tendríamos $s^{x'+1} = s^{x'+y+1}$. Además, está claro que el número y es el menor que cumple la condición, puesto que indica la longitud del ciclo (y empecemos por donde empecemos el ciclo, este siempre tiene la misma longitud). Así que $h = y$ si usamos la notación del Problema 4.2.7. Sin embargo, el

número x obtenido con este algoritmo no tiene por qué ser el mínimo que cumple $s^x = s^{x+y}$, ya que podría suceder, por ejemplo, que s^4 fuese igual que s^{4+10} y no lo habríamos comprobado porque al llegar al valor $y = 8$ cambiamos el valor de x de 4 a 8. Esto implica que x' tampoco tiene por qué ser el menor que cumple $s^{x'+1} = s^{x'+y+1}$. ¿Cómo podemos saber el m que buscamos entonces? Como ya hemos encontrado un x' que satisface la ecuación y sabemos quién es h, podemos usar la fuerza bruta probando si $s^{i+1} = s^{i+h+1}$ variando i. No es el método más eficaz, pero al menos como sabemos el valor de h reducirá bastante nuestras cuentas. Los detalles de los dos algoritmos que he usado están en la Sección A.4 del anexo.

Después de hacer 50 experimentos, he llegado a la conclusión de que la cota de 10^{10} dada por los autores es correcta, ya que la media obtenida ha sido de 10^{18} y sólo han salido 5 valores por debajo de 10^{10}. No he realizado más experimentos porque en cierto momento del experimento hay que trabajar con matrices de tamaño 12×12, cuyo tiempo de cálculo es bastante grande. Esto nos ha llevado a buscar alternativas, y hemos encontrado una posibilidad que todavía no hemos tenido tiempo de desarrollar completamente. Vamos a explicar ahora las ideas principales de esta alternativa.

Después de hacer la descomposición dada por el Teorema de Wedderburn-Artin, el semigrupo que nos interesa estudiar es el semigrupo multiplicativo de un anillo de matrices con entradas en $\mathbb{Z}_7$. Lo que se puede decir en este caso se podrá decir para cualquier anillo de matrices sobre un cuerpo finito. Por tanto, vamos a ponernos en esta situación más general, es decir, fijamos un entero positivo n y un cuerpo finito F con cardinal q, siendo q una potencia de un primo p, y S es el semigrupo multiplicativo del anillo de matrices $\mathcal{M}_n(F)$. Lo primero que vamos a hacer es calcular las longitudes de las colas y los ciclos de una matriz de Jordan elemental. Recordemos que estas matrices son las que tienen la forma:

$$J_n(a) = aI + N \quad \text{con} \quad N = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix}.$$

Lema 4.2.9. *La longitud de la cola de $J_n(0)$ es $n-1$ y la de su ciclo es 1. Si $a \neq 0$ entonces la longitud de la cola de $J_n(a)$ es 0 y la de su ciclo es el menor entero $h \geq n$ que es múltiplo del orden de a en F^* y tal que p divide a $\binom{h}{i}$ para todo $i = 1, 2 \dots, n-1$.*

Demostración. Obsérvese que, usando la fórmula del binomio de Newton,

$$\begin{aligned} J_n(a)^m &= \binom{m}{0} a^m I + \binom{m}{1} a^{m-1} N + \binom{m}{2} a^{m-2} N^2 + \cdots + \binom{m}{m-1} a N^{m-1} + \binom{m}{m} N^m \\ &= \begin{pmatrix} a^m & \binom{m}{1}a^{m-1} & \binom{m}{2}a^{m-2} & \binom{m}{3}a^{m-3} & \dots \\ 0 & a^m & \binom{m}{1}a^{m-1} & \binom{m}{2}a^{m-2} & \dots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & a^m & \binom{m}{1}a^{m-1} \\ 0 & 0 & \dots & 0 & a^m \end{pmatrix}. \end{aligned} \tag{4.2}$$

Si nos fijamos, esta matriz (4.2) está determinada por la fila superior que está formada por el vector

$$v_n(a,m) = \begin{cases} \left(a^m, \binom{m}{1}a^{m-1}, \dots, \binom{m}{m-1}a, 1, 0, \dots, 0\right) & \text{si } m < n \\ \\ \left(a^m, \binom{m}{1}a^{m-1}, \dots, \binom{m}{n-1}a^{m+1-n}\right) & \text{si } m \geq n \end{cases}$$

Por tanto, $J_n(a)^{m_1} = J_n(a)^{m_2}$ si y sólo si $v_n(a, m_1) = v_n(a, m_2)$. Si $a = 0$ entonces $v_n(a, m) = 0$ si $m \geq n$ y en caso contrario es el vector que tiene un 1 en la entrada $m + 1$. Veamos esto con un ejemplo, supongamos que estamos con $n = 4$, entonces:

$$\begin{aligned} v_4(0,1) =& (0,1,0,0) \\ v_4(0,2) =& (0,0,1,0) \\ v_4(0,3) =& (0,0,0,1) \\ v_4(0,4) =& (0,0,0,0) \\ & \dots \end{aligned}$$

Como se puede observar, con $m \geq 4$ tenemos que $v_4(0, m) = 0$. Visto esto, queda claro que si $a = 0$ entonces la longitud de la cola es $n - 1$ y la longitud del ciclo es 1.

Si $a \neq 0$ entonces $J_n(a)$ es invertible, con lo que la longitud de la cola es 0 y la del ciclo es el orden de $J_n(a)$, y este orden es el menor entero positivo m para el que $v_n(a, m) = (1, 0, \dots, 0)$. Si $m < n$, como la entrada $m+1$-ésima de $v_n(a, m)$ es 1, entonces el orden de $J_n(a)$ es al menos n. Si $m \geq n$, entonces la entrada i-ésima de $v_n(a, m)$ es $\binom{m}{i-1} a^{m-i+1}$ para todo $i = 1, \dots, n-1$. Por tanto, $J_n(a)^m = 1$ si y sólo si $m \geq n$, m es múltiplo del orden de a en F^* (para que la primera entrada de $v_n(a, m)$ sea 1) y p divide a $\binom{m}{i}$ para $i = 1, \dots, n-1$ (para que las demás entradas sean 0). □

Supongamos ahora que tenemos una matriz cualquiera A en $\mathcal{M}_n(F)$. Entonces, A tiene una forma canónina de Jordan formada por matrices de Jordan elementales $J_{n_1}(a_1), \dots, J_{n_k}(a_k)$, con $n_1 + n_2 + \dots + n_k = n$. Es decir, A es conjugada de la matriz

$$J = \begin{pmatrix} J_{n_1}(a_1) & & & \\ & J_{n_2}(a_2) & & \\ & & \ddots & \\ & & & J_{n_k}(a_k) \end{pmatrix}.$$

En otras palabras, $A = UJU^{-1}$ con U una matriz invertible. Por tanto, la longitud de la cola y del ciclo de A coincide con la de J. Además, podemos ver J como una matriz en el producto directo de los semigrupos multiplicativos de $M_{n_1}(L), \dots, M_{n_k}(L)$, donde L es un cuerpo que contiene todos los autovalores de A.

Ahora, uniendo los lemas 4.2.8 y 4.2.9, podemos obtener la siguiente proposición:

Proposición 4.2.10. *Sea $A \in \mathcal{M}_n(F)$ y sean $J_{n_1}(a_1), \dots, J_{n_k}(a_k)$ los bloques de la descomposición de Jordan de A. Si $a_i \neq 0$, sea h_i el orden multiplicativo de a_i, y sean $m_0 = \max\{n_i : a_i = 0\}$, $m_1 = \max\{n_i : a_i \neq 0\}$ y $h = \mathrm{mcm}\{h_i : a_i \neq 0\}$ (entendiendo que el máximo y el mínimo común múltiplo de un conjunto vacío de enteros positivos es 1). Entonces*

- *La longitud de la cola de A es $m_0 - 1$.*
- *La longitud del ciclo de A es el menor entero positivo $m \geq m_1$ que sea múltiplo de h y tal que p divide a $\binom{m}{i}$ para todo $i = 1, \dots, m_1 - 1$.*

Sin embargo, los autovalores de una matriz en $\mathcal{M}_n(F)$ pueden no estar en F, sino en un cuerpo más grande al que denotaremos L. En ese caso la forma canónica de Jordan estaría en $\mathcal{M}_n(L)$. Para evitar este problema consideraremos la forma canónica de A en $\mathcal{M}_n(F)$. Es decir, una matriz de la forma

$$C = \begin{pmatrix} C_{n_1}(P_1) & & & \\ & C_{n_2}(P_2) & & \\ & & \ddots & \\ & & & C_{n_k}(P_k) \end{pmatrix}, \tag{4.3}$$

donde $P_1, \ldots, P_k$ son polinomios mónicos irreducibles con coeficientes en F y si P es un polinomio de grado m entonces

$$C_n(P) = \begin{pmatrix} C(P) & I_m & 0 & 0 & \ldots & 0 \\ 0 & C(P) & I_m & \ldots & 0 & 0 \\ 0 & 0 & C(P) & \ddots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & C(P) & I_m \\ 0 & 0 & 0 & \ldots & 0 & C(P) \end{pmatrix},$$

y $C(P)$ es la matriz de compañía de P, es decir, si $P = X^m + a_{m-1}X^{m-1} + \ldots + a_1X + a_0$, entonces

$$C(P) = \begin{pmatrix} 0 & 0 & \ldots & 0 & -a_0 \\ 1 & 0 & \ldots & 0 & -a_1 \\ 0 & 1 & \ldots & 0 & -a_2 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & 1 & -a_{m-1} \end{pmatrix}.$$

Obsérvese que el polinomio característico de $C(P)$ es P y por tanto, si una matriz es conjugada de la matriz C de (4.3) entonces su polinomio característico es $P_1^{n_1} \cdots P_k^{n_k}$, con lo que $P_1, \ldots, P_k$ son los divisores irreducibles del polinomio característico de A. Además, si juntamos los P_i iguales, de forma que $Q_1, \ldots, Q_l$ son los diferentes P_i y $e_i = \text{máx}\{n_j : P_j = Q_i\}$, entonces $Q = Q_1^{e_1} \cdots Q_l^{e_l}$ es el polinomio mínimo de A, es decir el polinomio mónico de grado mínimo que cumple que $Q(A) = 0$.

Ahora podríamos calcular la longitud de la cola y el ciclo de A utilizando la forma canónica, en lugar de la de Jordan, porque en realidad la forma canónica determina la forma de Jordan. En efecto, si m es el grado de P, entonces cada bloque $C_n(P)$ de la forma canónica C de A da lugar a m matrices elementales de Jordan de tamaño n. Más concretamente, si $a_1, \ldots, a_m$ son las raíces de P, entonces los bloques a los que daría lugar $C_n(P)$ son las m matrices elementales de Jordan $J_n(a_1), \ldots, J_n(a_m)$.

Obsérvese que como sólo estamos considerando cuerpos finitos que son perfectos, es decir, todas sus extensiones finitas son de Galois, el polinomio irreducible P es separable, o sea no tiene raíces múltiples. Como además todos los elementos no nulos de un cuerpo finito son raíces de la unidad y como todas las raíces de P son conjugadas sobre K, todas tienen el mismo orden.

Teniendo en cuenta todo lo dicho hasta ahora tenemos la siguiente proposición.

Proposición 4.2.11. *Sea $A \in \mathcal{M}_n(F)$ y supongamos que su forma canónica es la matriz C de (4.3). Para cada $i = 1, \ldots, k$, sea a_i una raíz de P_i en una extensión de F. Si $a_i \neq 0$, sea h_i el orden multiplicativo de a y sean $m_0 = \text{máx}\{n_i : a_i = 0\}$, $m_1 = \text{máx}\{n_i : a_i \neq 0\}$ y $h = \text{mcm}\{\text{h}_\text{i} : \text{a}_\text{i} \neq 0\}$. Entonces*

1. *La longitud de la cola de A es $m_0 - 1$.*
2. *La longitud del ciclo de A es el menor entero positivo $m \geq m_1$ que sea múltiplo de h y tal que p divide a $\binom{m}{i}$ para todo $i = 1, \ldots, m_1 - 1$.*

Con todo lo visto arriba, podríamos generar un experimento para estimar las longitudes de la cola y el ciclo de muchos elementos de $M_2(\mathbb{Z}_7\mathcal{S}_5)$. Para ello utilizaríamos que

$$\mathcal{M}_2(\mathbb{Z}_7\mathcal{S}_5) \simeq \mathcal{M}_2(\mathbb{Z}_7) \times \mathcal{M}_2(\mathbb{Z}_7) \times \mathcal{M}_8(\mathbb{Z}_7) \times \mathcal{M}_8(\mathbb{Z}_7) \times \mathcal{M}_{10}(\mathbb{Z}_7) \times \mathcal{M}_{10}(\mathbb{Z}_7) \times \mathcal{M}_{12}(\mathbb{Z}_7),$$

con lo cual en lugar de generar elementos de $M_2(\mathbb{Z}_7\mathcal{S}_5)$, podríamos generar listas de matrices de tamaños $2, 2, 8, 8, 10, 10$ y 12 y calcular las longitudes de la cola y el ciclo de estas matrices mirando su forma canónica y utilizando lo que hemos visto antes. Sin embargo, calcular la forma canónica requiere factorizar el polinomio característico, entre otros cálculos, y eso lleva mucho tiempo. Por tanto, en lugar de generar matrices cualesquiera y a partir de ellas calcular su forma canónica lo que hacemos es generar al azar formas canónicas y, de hecho, esto lo podríamos hacer de forma algo virtual porque, en lugar de tomar al azar los polinomios irreducibles P_i lo que haríamos sería elegir un elemento a_i de una extensión apropiada de F y calcular su orden, de forma que el polinomio P_i sería el polinomio mínimo de a_i. Obsérvese que a_i es un elemento de $F[X]/(P_i)$ que es un cuerpo de cardinal q^{m_i} si P_i tiene grado m_i. Sin embargo, todavía nos queda bastante trabajo de análisis para poder realizar este experimento, ya que habría que comprender con qué probabilidad aparecería cada forma canónica.

4.2.4. Ataques al protocolo

Existen infinidad de métodos que intentan romper protocolos. Así que, dado un nuevo protocolo como el del artículo que estamos analizando, hay que comprobar que estos "ataques" no le sirvan de nada a Eva. Por tanto, los autores comprueban que algunos de los ataques más frecuentes son inútiles contra su protocolo.

Supongamos que Alicia y Bob se han puesto de acuerdo en una clave usando el protocolo que estamos estudiando y que Eva quiere obtener la clave. Para ello se tendría que enfrentar al *Problema de Diffie-Hellman* 3.6.1, pero ya hemos visto que ese problema se puede reducir al Problema del Logaritmo Discreto 3.1.1, así que los "ataques" que Eva va a intentar son los mismos que hemos visto para el Logaritmo Discreto.

Fuerza bruta

Este método no suele funcionar casi nunca, y el protocolo que estamos estudiando no es ninguna excepción. Si Eva intentara este método tendría que hacer como mucho tantas operaciones como elementos distintos genere la matriz escogida de $\mathcal{M}_s(\mathbb{Z}_n\mathcal{S}_m)$, y dado que el semigrupo tiene $n^{s^2 \cdot m!}$ elementos, seguramente Alicia y Bob habrán escogido una matriz en la que este número sea lo bastante grande como para que Eva se canse de hacer cuentas.

Paso de Niño-Paso de Gigante

En este caso, un gran problema es que el algoritmo tal y como lo hemos visto en la Sección 3.3 es para grupos, y nosotros tenemos un semigrupo. Pero a pesar de eso, si Eva quisiera usar este método tendría otro problema también muy importante que no va a ser el tiempo, sino la forma de "recordar" las matrices. Para llevar a cabo el método, se necesita una forma eficiente de almacenar matrices, por ejemplo se podría usar una *función Hash* o *función resumen* [27]. Aun así, las matrices que estamos tratando son demasiado complicadas y es difícil "resumirlas" de una forma útil.

Supongamos que estamos en $\mathcal{M}_2(\mathbb{Z}_7\mathcal{S}_5)$. Si recordamos, al principio de este capítulo, en la Sección 4.1, hemos calculado que para almacenar una matriz de tamaño 2×2 necesitamos 1440 *bits*. Por otro lado, en este algoritmo tenemos que almacenar $\sqrt{|\mathcal{M}_2(\mathbb{Z}_7\mathcal{S}_5)|} = \sqrt{7^{480}} \approx \sqrt{10^{406}} = 10^{203}$ matrices. Entonces, necesitaremos almacenar $1440 \cdot 10^{203} \approx 10^{207}$ *bits* para

poder realizar este algoritmo. Si tenemos en cuenta que $1\,TB = 8 \cdot 10^{12}$ *bits* [28], obtenemos que necesitaremos aproximadamente $10^{194}\,TB$ (o lo que es lo mismo, $10^{197}\,GB$). Estos números hacen que almacenar las matrices sea claramente infactible en cualquier ordenador. Sin embargo, alguien podría plantearse que no es necesario guardar todos los coeficientes, sino sólo los que sean distintos de cero. En ese caso, los autores del artículo que estamos estudiando afirman que como el número 10^{203} es muy grande, aún así no se podrían almacenar todas las matrices en la memoria del ordenador.

Algoritmo de Pohlig-Hellman

Para el algoritmo de Pohlig-Hellman visto en la Sección 3.4, los autores hacen sus argumentos sobre $\mathcal{M}_3(\mathbb{Z}_7\mathcal{S}_5)$. Los argumentos que dan son los siguientes:

1. Dada una matriz, no es sencillo saber su orden a pesar de saber el número de elementos de $\mathcal{M}_3(\mathbb{Z}_7\mathcal{S}_5)$, ya que no guardan ninguna relación.

2. En el paso 2.4.2 del algoritmo hay que resolver el Problema del Logaritmo Discreto en $\mathcal{M}_3(\mathbb{Z}_7\mathcal{S}_5)$ para calcular un entero l_j, lo cual según los autores sólo puede hacerse mediante el método de la fuerza bruta que ya hemos visto que no es recomendable. Es cierto que el l_j que estaremos buscando será bastante menor que el entero x tal que $a^x = b$, sin embargo, los autores dicen que como el tamaño del semigrupo es 7^{1080}, a pesar de esto será muy difícil que Eva consiga calcular l_j.

Algoritmo ρ de Pollard

Si Eva intentara usar el algoritmo ρ de Pollard con la matriz $M \in \mathcal{M}_3(\mathbb{Z}_7\mathcal{S}_5)$ escogida por Alicia y Bob, antes de empezar a hacer cuentas ya tendría un problema: el algoritmo tal y como lo hemos visto en la Sección 3.5 es para grupos, y $\mathcal{M}_3(\mathbb{Z}_7\mathcal{S}_5)$ no lo es. Si a pesar de esto, Eva quisiera hacer este algoritmo tendría, como es lógico, varios problemas. El primero de ellos es que es necesario conocer el orden del subgrupo generado por la matriz M, pero por lo general Eva no lo conocerá y según los autores lo tendrá bastante difícil para calcularlo. Supongamos que lo consigue calcular y que trata después de usar el algoritmo para encontrar un a tal que $M^a = N$. Entonces el algoritmo ρ de Pollard se basará en encontrar unos coeficientes enteros $\alpha_i, \alpha_j, \beta_i, \beta_j$ tales que

$$M^{\alpha_i} N^{\beta_i} = M^{\alpha_j} N^{\beta_j}.$$

Después, usando que $M^a = N$, Eva obtendría:

$$M^{\alpha_i} M^{a\beta_i} = M^{\alpha_j} M^{a\beta_j}.$$

El siguiente paso del algoritmo sería llegar a una expresión del tipo:

$$M^{a(\beta_i - \beta_j)} = M^{\alpha_i - \alpha_j}.$$

Sin embargo, aquí surge otro problema, si la matriz M no es invertible Eva no podrá haber llegado a este paso, y en el semigrupo $\mathcal{M}_s(\mathbb{Z}_7\mathcal{S}_5)$ no todas las matrices son invertibles.

Debido a estos motivos, los autores Kahrobaei, Koupparis y Shpilrain concluyen que este método (al menos en su forma clásica explicada en la Sección 3.5) no sirve para romper su protocolo.

Existe una versión para semigrupos de este algoritmo que puede ayudar a Eva a romper el código. Sin embargo, en ella es esencial saber calcular el ciclo de un elemento de forma eficiente, lo cual puede ser de nuevo un problema (algunos ejemplos de cómo intentar calcular el ciclo de

un elemento están en 4.2.3). El algoritmo es el siguiente: supongamos que dados un semigrupo S y $a, b \in S$ queremos encontrar un entero x tal que $a^x = b$ y que sabemos que el ciclo de a es h. Entonces el cálculo de los α_i y los β_i será igual que en el algoritmo original cambiando la palabra "grupo" por "semigrupo" y n por h. Al llegar al momento de tener que resolver $a^{\alpha_i}a^{x\beta_i} = a^{\alpha_j}a^{x\beta_j}$, podemos fijarnos en que es equivalente a resolver $a^{\alpha_i+x\beta_i} = a^{\alpha_j+x\beta_j}$. Y resolver esto último es lo mismo que resolver la ecuación de congruencias $\alpha_i + x\beta_i \equiv \alpha_j + x\beta_j$ mód h que es equivalente a la ecuación $x(\beta_i - \beta_j) \equiv \alpha_j - \alpha_i \mod h$. A partir de aquí se trata, como en el algoritmo original, de encontrar una solución entera para la ecuación $xr \equiv u \mod h$ con $r = \beta_i - \beta_j$ y $u = \alpha_i - \alpha_j$.

4.3. Conclusiones del protocolo

A pesar de las comprobaciones hechas por Kahrobaei, Koupparis y Shpilrain sobre la eficacia del protocolo y reproducidas por mí en este trabajo, eso no quiere decir que el protocolo sea infalible cien por cien. Por eso, los autores escriben en su artículo varias frases como "no se puede resolver al menos con esta forma estándar del algoritmo", puesto que saben que podría suceder que alguien encontrara una forma de romper su código que ellos no hayan pensado (esto podría ocurrir, por ejemplo, con la versión del algoritmo ρ de Pollard para semigrupos y que ellos no han tenido en cuenta). Incluso proponen un reto dando unas matrices M, M^a y M^b para ver si alguien consigue encontrar los valores a y b. De hecho, recientemente, en julio de este año 2015, Chris Monico y Mara Dicle Neusel (1964-2014) han publicado un artículo llamado *Cryptanalysis of a system using matrices over group rings* [17] en el que explican que el protocolo propuesto por D. Kahrobael, C. Koupparis y V. Shpilrain se puede romper "fácilmente" y han resuelto el reto propuesto.

Apéndice A

Códigos para GAP

En este anexo están los códigos de GAP que he ido haciendo y usando a lo largo del trabajo. Como GAP era un software completamente nuevo para mí, las referencias usadas en esta parte son prácticamente manuales de referencia: [7], [8], [11] y [25].

Observación A.0.1. Por defecto, GAP multiplica las permutaciones al revés que nosotros, es decir, si queremos multiplicar entre sí las permutaciones a y b de la forma $a \cdot b$, para que nuestras cuentas "a mano" coincidan con las de GAP, en el software tendremos que escribir $b \cdot a$. A pesar de esto, como nosotros vamos a trabajar con elementos aleatorios, en el fondo no nos va a afectar en los experimentos.

A.1. Para Iniciar

En esta sección, los apartados *Primera forma*, *Segunda forma* y *Tercera forma* (A.1.1, A.1.2 y A.1.3 respectivamente) contienen las funciones necesarias para realizar los experimentos como indican las tres formas explicadas en 4.2.1.

A.1.1. Primera forma

La primera forma de realizar los experimentos es usando todas las funciones de GAP para multiplicar elementos de $\mathcal{M}_s(\mathbb{Z}_n\mathcal{S}_m)$ sin añadir ninguna. Para ello, primero tengo que definir unas variables globales que usaré a menudo.

Variables globales

```
n := 0;
m := 0;
ZN := 0;
SM := 0;
ZNSM := 0;
emb := 0;
zero := 0;
e := 1;
Lperm := 0;
Lent := 0;
longLperm := 0;
```

La siguiente función inicializa $\mathbb{Z}_n$, $\mathcal{S}_m$ y $\mathbb{Z}_n\mathcal{S}_m$ según dónde se quiera trabajar. Además, también inicializa otras variables útiles.

Función Iniciar

```
Iniciar := function(ene,eme)
      n := ene;
      m := eme;
      ZN := GF(n);
      SM := SymmetricGroup(m);
      ZNSM := GroupRing(ZN,SM);
      emb := Embedding(SM,ZNSM);
      Lperm := Elements(SM);
      Lent := Elements(ZN);
      longLperm := Length(Lperm);
      zero := 0*(Lperm[1]^emb);
      e := ()^emb;
end;
```

Las dos siguientes funciones crean una lista de ceros (ceros enteros) y una matriz de ceros (ceros enteros).

Función CreaListas

```
CreaListas := function(tamano)
      local lista, i;
      lista := [0];
      for i in [2 .. tamano] do
            Add(lista,0);
      od;
      return lista;
end;
```

Función CreaMatrices

```
CreaMatrices := function(ene,eme)
      local lista, i;
      lista := [CreaListas(eme)];
      for i in [2 .. ene] do
            Add(lista,CreaListas(eme));
      od;
      return lista;
end;
```

A.1.2. Segunda forma

La segunda forma de realizar los experimentos es usando la forma de multiplicar elementos de $\mathcal{M}_s(\mathbb{Z}_n\mathcal{S}_m)$ de GAP pero usando mi función *Potencia*. En este caso, las variables globales y la función *Iniciar* son las mismas que en la primera forma.

Estas son las funciones *CBase* y *Potencia* explicadas en el Ejemplo 2.2.16. Observar que la función *Potencia* sirve también para matrices.

Función CBase

```
CBase := function(base,num)
    local lista, numero;
    numero := num;
    lista := [numero mod base];
    numero := Int(numero/base);
    while numero < > 0 do
        Add(lista,numero mod base);
        numero := Int(numero/base);
    od;
    return lista;
end;
```

Función Potencia

```
Potencia := function(g,a)
    local resultado, potencias, fact, long, i;
    resultado := 1;
    potencias := g;
    fact := CBase(2,a);
    long := Length(fact);
    for i in [1 .. long] do
        if fact[i] = 1 then
            resultado := resultado*potencias;
        fi;
        potencias := potencias*potencias;
    od;
    return resultado;
end;
```

A.1.3. Tercera forma

Por último, la tercera forma de realizar los experimentos es usando la manera de multiplicar elementos de $\mathcal{M}_s(\mathbb{Z}_n\mathcal{S}_m)$ explicada en el artículo de Kahrobaei, Koupparis y Shpilrain (es decir, precalculando las multiplicaciones de los elementos de $\mathcal{S}_5$) y usando mi función *Potencia*. En este caso, las funciones *CBase*, *CreaListas* y *CreaMatrices* son las mismas que en la primera forma A.1.1 y en la segunda forma A.1.2.

Las variables globales en este caso son las siguientes.

Variables globales

```
n := 0;
m := 0;
ZN := 0;
SM := 0;
ZNSM := 0;
emb := 0;
zero := 0;
e := 1;
Lperm := 0;
Lent := 0;
longLperm := 0;
tabla := 0;
```

En esta tercera forma, la función *Iniciar* es ligeramente distinta.

Función Iniciar

```
Iniciar := function(ene,eme)
      local i, j;
      n := ene;
      m := eme;
      ZN := GF(n);
      SM := SymmetricGroup(m);
      ZNSM := GroupRing(ZN,SM);
      emb := Embedding(SM,ZNSM);
      Lperm := Elements(SM);
      Lent := Elements(ZN);
      longLperm := Length(Lperm);
      zero := 0*(Lperm[1]^emb);
      e := ()^emb;
      tabla := CreaMatrices(longLperm,longLperm);
      for i in [1 .. longLperm] do
            for j in [1 .. longLperm] do
                  tabla[i][j] := Lperm[j]*Lperm[i];
            od;
      od;
end;
```

Ahora, pasemos al método de *cuadrar y multiplicar* del Ejemplo 2.2.16 para matrices mediante la tercera forma, el cual lo he ido programando poco a poco siguiendo los siguientes pasos:

1. La multiplicación entre elementos de $\mathcal{S}_m$ usando la tabla precalculada.

2. La multiplicación entre elementos de $\mathbb{Z}_n\mathcal{S}_m$.

3. La multiplicación entre matrices con entradas en $\mathbb{Z}_n\mathcal{S}_m$.

4. Crear una matriz identidad en las matrices sobre $\mathbb{Z}_n\mathcal{S}_m$.

5. El método de cuadrar y multiplicar.

(1) Función MultSM

```
MultSM := function(a,b)
      local posa, posb, res;
      posa := Position(Lperm,a);
      posb := Position(Lperm,b);
      return tabla[posa][posb];
end;
```

(2) Función MultZNSM

```
MultZNSM := function(a,b)
    local res, i, j, facta, factb, coefa, coefb;
    facta := Support(a);
    coefa := CoefficientsBySupport(a);
    factb := Support(b);
    coefb := CoefficientsBySupport(b);
    res := zero;
    for i in [1 .. Length(facta)] do
        for j in [1 .. Length(factb)] do
            res := res + (coefa[i]*coefb[j]*(MultSM(facta[i],factb[j])^emb));
        od;
    od;
    return res;
end;
```

(3) Función MultMat

```
MultMat := function(M,N)
    local R, dim, i, j, k;
    dim := Length(M);
    R := CreaMatrices(dim, dim);
    for i in [1 .. dim] do
        for j in [1 .. dim] do
            R[i][j] := zero;
        od;
    od;
    for i in [1 .. dim] do
        for j in [1 .. dim] do
            for k in [1 .. dim] do
                R[i][j] := R[i][j] + MultZNSM(M[i][k],N[k][j] );
            od;
        od;
    od;
    return R;
end;
```

(4) Función Identidad

```
Identidad := function(tamano)
    local M, i, j;
    M := CreaMatrices(tamano,tamano);
    for i in [1 .. tamano] do
        for j in [1 .. tamano] do
            if i = j then
                M[i][j] := e;
            else
                M[i][j] := zero;
            fi;
        od;
    od;
    return M;
end;
```

(5) Función PotenciaMat

```
PotenciaMat := function(M,a)
    local resultado, potencias, fact, long, i;
    resultado := Identidad(Length(M));
    potencias := M;
    fact := CBase(2,a);
    long := Length(fact);
    for i in [1 .. long] do
        if fact[i] = 1 then
            resultado := MultMat(resultado,potencias);
        fi;
        potencias := MultMat(potencias,potencias);
    od;
    return resultado;
end;
```

A.2. Velocidad

En esta sección voy a poner los códigos usados para hacer el experimento de velocidad de las tres formas explicadas en 4.2.1. En estos experimentos, la entrada s indica el tamaño de las matrices aleatorias que queremos usar, *exp* indica el exponente al que queremos elevar cada matriz y *num* indica para cuántas matrices queremos repetir esto. Por otro lado, para ejecutar cada una de las formas en GAP necesitaremos usar sus funciones respectivas del apartado anterior A.1.

Función Speed forma 1

```
Speed1 := function(s,exp,num)
    local M, i, t0, tTotal, res;
    tTotal := 0;
    for i in [1 .. num] do
        M := RandomMat(s,s,ZNSM);
        t0 := Runtime();
        res := M^exp;
        tTotal := tTotal + (Runtime() - t0);
    od;
    return Float((tTotal/num)/1000);
end;
```

Función Speed forma 2

```
Speed2 := function(s,exp,num)
    local M, i, t0, tTotal;
    tTotal := 0;
    for i in [1 .. num] do
        M := RandomMat(s,s,ZNSM);
        t0 := Runtime();
        Potencia(M,exp);
        tTotal := tTotal + (Runtime() - t0);
    od;
    return Float((tTotal/num)/1000);
end;
```

Función Speed forma 3

```
Speed3 := function(s,exp,num)
    local M, i, t0, tTotal;
    tTotal := 0;
    for i in [1 .. num] do
        M := RandomMat(s,s,ZNSM);
        t0 := Runtime();
        PotenciaMat(M,exp);
        tTotal := tTotal + (Runtime() - t0);
    od;
    return Float((tTotal/num)/1000);
end;
```

A.3. Hipótesis de decisión

En esta sección están los códigos usados para realizar los experimentos 1, 2 y 3 explicados en la Sección 4.2.2.

A.3.1. Experimento 1

Vamos a ver los códigos usados para el primer experimento. Para ello, veremos tres funciones de entre las cuales las dos primeras tienen elementos en común.

- Ambas necesitan el tamaño de las matrices que queremos usar (s), el número de veces que queremos que se realice el experimento antes de devolvernos un resultado (num) y qué entrada de las matrices queremos que mire el algoritmo para darnos el resultado (i, j). Necesita la entrada porque es más fácil realizar el experimento entrada por entrada que con las cuatro entradas de las matrices al mismo tiempo.
- Las dos funciones devuelven un array en el que cada posición del array representa una permutación (en $\mathbb{Z}_7\mathcal{S}_5$ el array tendrá tamaño 120).
- En el array que devuelven las dos funciones, el número (entero) x que haya en la posición i indicará que al mirar la entrada que le hayamos indicado de las matrices, la permutación i ha salido x veces (sin tener en cuenta los coeficientes).

La primera función calcula cuántas veces aparece cada permutación en matrices aleatorias elevadas a $a \cdot b$ con $a, b \in [10^6, 10^8]$ escogidos de forma uniformemente aleatoria.

Función Experimento 1 (modo $a \cdot b$)

```
CuantasPerAB := function(s,num,i,j)
    local M, k, x, l, z, lista, listaF, a, b, Mab ;
    listaF := CreaListas(longLperm);
    for k in [1 .. num] do
        M := RandomMat(s,s,ZNSM);
        a := Random([10^6 .. 10^8]);
        b := Random([10^6 .. 10^8]);
        Mab := M^(a*b);
        x := Mab[i][j];
        lista := Support(x);
        for l in [1 .. Length(lista)] do
            z := Position(Lperm, lista[l]);
            listaF[z] := listaF[z] + 1;
        od;
    od;
    return listaF;
end;
```

La segunda función calcula cuántas veces aparece cada permutación en matrices aleatorias elevadas a c con $c \in [10^{12}, 10^{16}]$ escogido de forma uniformemente aleatoria.

Función Experimento 1 (modo c)

```
CuantasPerC := function(s,num,i,j)
    local M, k, x, l, z, lista, listaF, c, Mc ;
    listaF := CreaListas(longLperm);
    for k in [1 .. num] do
        M := RandomMat(s,s,ZNSM);
        c := Random([10^12 .. 10^16]);
        Mc := M^c;
        x := Mc[i][j];
        lista := Support(x);
        for l in [1 .. Length(lista)] do
            z := Position(Lperm, lista[l]);
            listaF[z] := listaF[z] + 1;
        od;
    od;
    return listaF;
end;
```

La siguiente función sirve para obtener los valores que necesito para las gráficas Q-Q. En ella, x es el vector de datos para el cual se quiere calcular F^{-1} (siendo F la función de distribución acumulada de los datos) y *cant* es en cuántos valores comprendidos entre 0 y 1 queremos que se evalúe la función F^{-1}.

Función cdfInversa

```
cdfInversa := function(x,cant)
    local tot,long,F,res,i,j,k,l;
    tot := 0;
    long := Length(x);
    F := CreaListas(long);
    res := CreaListas(cant);
    for i in [1 .. long] do
        tot := tot + x[i];
    od;
    F[1] := Float(x[1]/tot);
    for j in [2 .. long ] do
        F[j] := F[j-1] + Float(x[j]/tot);
    od;
    for k in [1 .. cant] do
        for l in [1 .. long] do
            if F[l] < Float(k/cant) then
                res[k] := l;
            fi;
        od;
    od;
    return res;
end;
```

Una vez obtenidos los datos dados por la función *cdfInversa* al usar como entrada los datos de las dos distribuciones que quiero comparar, he usado el programa R-Studio [22] para dibujar las gráficas de la Figura 4.2 (también he usado el programa R-studio para dibujar las gráficas de la Figura 4.3 del experimento 2).

A.3.2. Experimento 2

En esta segunda sección están los códigos que he usado para realizar el experimento 2. Las tres funciones que hay en esta sección tienen varios elementos en común con las de la Sección A.3.1. En particular, las dos primeras tienen en común la entrada y la salida.

La primera función calcula cuántas veces aparece cada permutación en matrices aleatorias elevadas a un número entero $a \in [10^{12}, 10^{18}]$ elegido de forma uniformemente aleatoria.

Función Experimento 2 (modo a)

```
CuantasPerA := function(s,num,i,j)
    local M, k, x, l, z, lista, listaF, a, Ma ;
    listaF := CreaListas(longLperm);
    for k in [1 .. num] do
        M := RandomMat(s,s,ZNSM);
        a := Random([10^12 .. 10^16]);
        Ma := M^a;
        x := Ma[i][j];
        lista := Support(x);
        for l in [1 .. Length(lista)] do
            z := Position(Lperm, lista[l]);
            listaF[z] := listaF[z] + 1;
        od;
    od;
    return listaF;
end;
```

La segunda función calcula cuántas veces aparece cada permutación en matrices aleatorias de $\mathbb{Z}_7\mathcal{S}_5$.

Función Experimento 2 (modo sin elevar)

```
CuantasPer := function(s,num,i,j)
    local M, k, x, l, z, lista, listaF;
    listaF := CreaListas(longLperm);
    for k in [1 .. num] do
        M := RandomMat(s,s,ZNSM);
        x := M[i][j];
        lista := Support(x);
        for l in [1 .. Length(lista)] do
            z := Position(Lperm, lista[l]);
            listaF[z] := listaF[z] + 1;
        od;
    od;
    return listaF;
end;
```

La tercera función es la función *cdfInversa* ya explicada en A.3.1.

A.3.3. Experimento 3

Por último, en esta sección vamos a ver el código necesario para realizar el tercer experimento. En este caso necesitaré cinco funciones.

La primera función crea una lista de tamaño n^3 (suponiendo que estemos en $\mathbb{Z}_n\mathcal{S}_m$) en la que están todas las posibles combinaciones de elementos de $\mathbb{Z}_n$. Por ejemplo, en $\mathbb{Z}_2$, la lista será:

$$\{(0,0,0),(0,0,1),(0,1,0),(0,1,1),(1,0,0),(1,0,1),(1,1,0),(1,1,1)\}.$$

Esta función la necesitaremos para poder hacer funcionar correctamente las otras cuatro funciones.

Función CreaLaTabla

```
CreaLaTabla := function()
    local lista, listaF, i, j, k;
    listaF := [[0*Z(n),0*Z(n),0*Z(n)]];
    for i in [1 .. n] do
        for j in [1 .. n] do
            for k in [1 .. n] do
                if i <> 1 then
                    lista := [Lent[i],Lent[j],Lent[k]];
                    Add(listaF,lista);
                else
                    if j <> 1 then
                        lista := [Lent[i],Lent[j],Lent[k]];
                        Add(listaF,lista);
                    else
                        if k <> 1 then
                            lista := [Lent[i],Lent[j],Lent[k]];
                            Add(listaF,lista);
                        fi;
                    fi;
                fi;
            od;
        od;
    od;
    return listaF;
end;
```

Las siguientes cuatro funciones cumple las siguientes propiedades:

- Como en las funciones del experimento 1, necesitan el tamaño de las matrices que queremos usar (s), el número de veces que queremos que se realice el experimento antes de devolvernos un resultado (num) y qué entrada de las matrices queremos que mire el algoritmo para darnos el resultado (i, j).

- Las funciones devuelven una tabla en la que la posición i, j representa la combinación i de elementos de $\mathbb{Z}_n$ en la lista dada por la función anterior (función *CreaLaTabla*) y la permutación j de las permutaciones de $\mathcal{S}_m$ (en $\mathbb{Z}_7\mathcal{S}_5$ la tabla tendrá tamaño $7^3 \times 120$).

- La tabla que devuelve cada una de las funciones se "rellena" de la siguiente forma: en un paso, después de hacer los cálculos que correspondan se llega a una tripla con tres matrices. Entonces, miramos la entrada que nos interese en cada una de las matrices de la tripla (miramos la misma entrada en las tres matrices) y apuntamos para cada una de las permutaciones de $\mathcal{S}_m$, la combinación de coeficientes que aparecen en la tripla. Después, si por ejemplo hemos mirado la permutación j de $\mathcal{S}_m$ y nos ha salido la combinación de coeficientes i de la lista dada por la primera función de este experimento (función *CreaLaTabla*), anotaremos un $+1$ en la posición i, j de la tabla final. En la parte del

trabajo en la que explico el experimento 3 hay un ejemplo que muestra cómo rellenar la tabla de esta forma.

Vamos a ver ahora el algoritmo de estas funciones. El siguiente algoritmo *TriplasAB* "mira" las triplas de la forma (M^a, M^b, M^{ab}) con a, b enteros aleatorios escogidos de forma uniforme en el intervalo $[10, 20]$ y M una matriz aleatoria de $\mathcal{M}_s(\mathbb{Z}_n\mathcal{S}_m)$. También he usado otros dos algoritmos que estudian las triplas de la forma (M^a, M^b, M^{a+b}) y las de la forma (M^a, M^b, M^c) con M una matriz aleatoria y $a, b \in [10, \ldots, 20]$ y $c \in [20, \ldots, 40]$ escogidos de forma uniformemente aleatoria. Estos dos últimos algoritmos no los he incluido porque simplemente hay que hacer unas pequeñas modificaciones en el algoritmo *TriplasAB*: tan sólo hay que cambiar $Mab := Ma\hat{}b$ por $Mab := M\hat{}(a+b)$ y $Mab := M\hat{}c$, respectivamente (en el caso del último habría que definir además la variable c).

Función Experimento 3 (modo ab)

```
TriplasAB := function(s,num,i,j)
    local res, M, a, b, Ma, Mb, Mab, xa, xb, xab, k, fact1, fact2, fact3, coef1,
coef2, coef3, posicion, x, comp, y, z, l, m;
    res := CreaMatrices(n^3,longLperm);
    for k in [1 .. num] do
        M := RandomMat(s,s,ZNSM);
        a :=Random([10 .. 20]);
        b :=Random([10 .. 20]);
        Ma := M^a;
        Mb := M^b;
        Mab := Ma^b;
        xa := Ma[i][j];
        xb := Mb[i][j];
        xab := Mab[i][j];
        fact1 := Support(xa);
        fact2 := Support(xb);
        fact3 := Support(xab);
        coef1 := CoefficientsBySupport(xa);
        coef2 := CoefficientsBySupport(xb);
        coef3 := CoefficientsBySupport(xab);
        for l in [1 .. longLperm] do
            if (Lperm[l] in fact1) then
                x := coef1[Position(fact1,Lperm[l])];
            else
                x := 0*Z(n);
            fi;
            if (Lperm[l] in fact2) then
                y := coef2[Position(fact2,Lperm[l])];
            else
                y := 0*Z(n);
            fi;
```

```
                if (Lperm[l] in fact3) then
                    z := coef3[Position(fact3,Lperm[l])];
                else
                    z := 0*Z(n);
                fi;
                m := [x,y,z];
                comp := CreaLaTabla();
                res[Position(comp,m)][l] := res[Position(comp,m)][l] + 1;
            od;
        od;
        return res;
end;
```

Por último, este algoritmo estudia las triplas de la forma (M_1, M_2, M_3) con M_1, M_2 y M_3 matrices aleatorias de $\mathcal{M}_s(\mathbb{Z}_n\mathcal{S}_m)$.

Función Experimento 3 (modo "sin elevar")

```
Triplas := function(s,num,i,j)
    local res,M1,M2,M3,x1,x2,x3,k,fact1,fact2,fact3,coef1,coef2,coef3,
posicion,x,comp,y,z,l,m;
    res := CreaMatrices(n^3,longLperm);
    for k in [1 .. num] do
        M1 := RandomMat(s,s,ZNSM);
        M2 := RandomMat(s,s,ZNSM);
        M3 := RandomMat(s,s,ZNSM);
        x1 := M1[i][j];
        x2 := M2[i][j];
        x3 := M3[i][j];
        fact1 := Support(x1);
        fact2 := Support(x2);
        fact3 := Support(x3);
        coef1 := CoefficientsBySupport(x1);
        coef2 := CoefficientsBySupport(x2);
        coef3 := CoefficientsBySupport(x3);
        for l in [1 .. longLperm] do
            if (Lperm[l] in fact1) then
                x := coef1[Position(fact1,Lperm[l])];
            else
                x := 0*Z(n);
            fi;
            if (Lperm[l] in fact2) then
                y := coef2[Position(fact2,Lperm[l])];
            else
                y := 0*Z(n);
            fi;
```

```
                if (Lperm[l] in fact3) then
                    z := coef3[Position(fact3,Lperm[l])];
                else
                    z := 0*Z(n);
                fi;
                m := [x,y,z];
                comp := CreaLaTabla();
                res[Position(comp,m)][l] := res[Position(comp,m)][l] + 1;
            od;
        od;
        return res;
end;
```

A.4. Existencia de ciclos

En este apartado voy a poner los códigos de GAP referentes al experimento realizado para ver si es poco probable que existan ciclos o no 4.2.3. Los códigos están pensados para el problema concreto que tengo que tratar, es decir, con matrices. Si deseáramos usar dichos algoritmos con números enteros habría que hacer unas pequeñas variaciones.

La primera función nos devuelve unos enteros m' y h (en ese orden) tales que $M^{m'} = M^{m'+h}$. Sin embargo, el entero m' no tiene por qué ser el menor que lo cumple. Como no sé una cota superior de los cálculos, he hecho que el usuario tenga que introducir un número máximo de iteraciones que desea; si dicho número de iteraciones se supera el algoritmo devolverá $m' = -1$ y $h = -1$ para que el usuario sepa que no ha encontrado ninguna órbita.

Función EncuentraOrbitasMat

```
EncuentraOrbitasMat := function(M,max)
    local a,b,m,h,x,y,i;
    if M^0 = M then
        return [0,1];
    fi;
    a := 1;
    b := 2;
    m := 1;
    h := 1;
    x := M;
    y := M*x;
    for i in [1 .. max] do
        if x = y then
            m := a;
            return [m,h];
        fi;
```

```
        if b = 2*a then
            a := b;
            b := a + 1;
            x := y;
            y := M*x;
            h := 1;
        else
            b := b + 1;
            h := h + 1;
            y := M*y;
        fi;
    od;
    return [-1,-1];
end;
```

La segunda función encuentra, ahora sí, el menor entero m que cumple $M^{m+1} = M^{m+h+1}$ usando el h dado por el método anterior. De nuevo, he incluido un parámetro de iteraciones máximas que si el algoritmo supera sin encontrar ninguna órbita devolverá el valor $m = -1$ para que el usuario sepa que no ha encontrado solución.

Función EncuentraMMat

```
EncuentraMMat := function(M,h,max)
    local m, x, y, i, j;
    m := 0;
    x := M;
    y := M;
    for j in [1 .. h] do
        y := y*M;
    od;
    for i in [1 .. max] do
        if x = y then
            return m;
        fi;
        x := x*M;
        y := y*M;
        m := m + 1;
    od;
    return -1;
end;
```

Bibliografía

[1] Ángel Ángel, J. J.: *Criptografía Para Principiantes.*
Disponible en www.math.com.mx/criptografia.html

[2] Aspnes, J.: *Notes on Randomized Algorithms.* Yale University, 2014.

[3] Boneh, D.: *The Decision Diffie-Hellman Problem.* ANTS 1998, 48-63.

[4] Busqué Roca, C. - Saorín Castaño, M. - Simón Pinero, J. J.: *Curso de conjuntos y números. Guiones de clase.* Universidad de Murcia, Facultad de Matemáticas, 2011.

[5] Computer Security Resource Center NIST: *Data Encryption Standard.*
Disponible en http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf

[6] Diffie, W. - Hellman, M. E.: *New Directions in Cryptography.* IEEE Transactions on Information Theory IT-22, 1976, 644-654.

[7] GAP System for Computational Discrete Algebra: *GAP - A Tutorial.*
Disponible en http://www.gap-system.org/Manuals/doc/tut/chap0.html

[8] GAP System for Computational Discrete Algebra: *GAP - Reference Manual.*
Disponible en http://www.gap-system.org/Manuals/doc/ref/chap0.html

[9] GAP System for Computational Discrete Algebra: *Programa GAP.*
Disponible en http://www.gap-system.org

[10] Gómez, J.: *Matemáticos, espías y piratas informáticos. Codificación y criptografía.* RBA Colección El mundo es matemático, 2010.

[11] Hulpke, A.: *Abstract Algebra in GAP.* Colorado State University, Department of Mathematics, 2011.

[12] Information Security Stack Exchange: *Rho.*
Disponible en http://security.stackexchange.com/questions/10659/general-purpose-slow-unique-hash-routine-for-dup-checking-of-private-data-witho

[13] Kahrobaei, D. - Koupparis, C. - Shpilrain, V.: *Public Key Exchange Using Matrices Over Group Rings.* Groups, Complexity, and Cryptology 5, 2013, 97–115.

[14] Koblitz, N.: *A Course in Number Theory and Cryptography.* Springer Graduate Texts in Mathematics, 1994.

[15] Martínez Hernández, J.: *Álgebra Conmutativa.* Universidad de Murcia, Facultad de Matemáticas, curso 2014-2015.

[16] Menezes, A. - Van Oorschot, P. - Vanstone, S.: *Handbook of Applied Cryptography*. CRC Press, 1996.

[17] Monico, C. - Neusel, M. D.: *Cryptanalysis of a system using matrices over group rings*. Texas Tech University, Department of Mathematics and Statistics, 2015. Preprint.

[18] Pierce, R. S.: *Associative Algebras*. Springer Graduate Texts in Mathematics, 1982.

[19] Pollard, J. M.: *Monte Carlo Methods for Index Computation mod p*. Mathematics of Computation, Volume 32, Number 143, 1978, 918-924.

[20] Del Río Mateos, A.: *Introducción a la Criptografía Matemática*. Universidad de Murcia, Facultad de Matemáticas, 2013.

[21] Del Río Mateos, A. - Simón Pinero, J. J. - Del Valle Robles, A.: *Álgebra Básica*. Universidad de Murcia, Facultad de Matemáticas, 2001.

[22] RStudio: *Programa RStudio*.

Disponible en https://www.rstudio.com

[23] Santamaría Fernández, J.: *El logaritmo discreto y sus aplicaciones en criptografía*. Proyecto Final de Carrera. Universidad de Cantabria, 2013. Otras responsabilidades: Sadornil Renedo, D.

Disponible en http://repositorio.unican.es/xmlui/bitstream/handle/10902/3101/Jennifer%20Santamaria%20Fernandez.pdf?sequence=1

[24] Saorín Castaño, M.: *Ecuaciones Algebraicas*. Universidad de Murcia, Facultad de Matemáticas, curso 2013-2014.

[25] Vendramin, L.: *Una introducción al álgebra con GAP*. Universidad de Chile, Santiago de Chile, 2014.

[26] Wikipedia, la enciclopedia libre: *Criptografía asimétrica*.

Disponible en https://en.wikipedia.org/wiki/Public-key_cryptography

[27] Wikipedia, la enciclopedia libre: *Función hash*.

Disponible en https://en.wikipedia.org/wiki/Hash_function

[28] Wikipedia, la enciclopedia libre: *Terabyte*.

Disponible en https://es.wikipedia.org/wiki/Terabyte

[29] Zoroa Alonso, N. - Zoroa Terol, P.: *Elementos de Probabilidades*. Diego Marín, 2008.