UNIVERSIDAD DE GRANADA

Facultad de Ciencias y Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Doble Grado en Ingeniería Informática y Matemáticas

TRABAJO DE FIN DE GRADO

Criptosistemas basados en el problema de la mochila

Presentado por:
Juan Manuel Mateos Pérez

Tutores:
Gabriel Navarro Garulo
Departamento de Ciencias de la Computación e Inteligencia Artificial

Francisco Javier Lobillo Borrero
Departamento de Álgebra

Curso académico 2023-2024

Criptosistemas basados en el problema de la mochila

Juan Manuel Mateos Pérez

Juan Manuel Mateos Pérez *Criptosistemas basados en el problema de la mochila*.

Trabajo de fin de Grado. Curso académico 2023-2024.

Responsables de tutorización

Gabriel Navarro Garulo
Departamento de Ciencias de la Computación e Inteligencia Artificial

Francisco Javier Lobillo Borrero
Departamento de Álgebra

Doble Grado en Ingeniería Informática y Matemáticas

Facultad de Ciencias y Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Universidad de Granada

Declaración de originalidad

D. Juan Manuel Mateos Pérez

Declara explícitamente que el trabajo presentado como Trabajo de Fin de Grado (TFG), correspondiente al curso académico 2023-2024, es original, en el sentido de que no ha utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Granada a 4 de diciembre de 2023

Fdo: Juan Manuel Mateos Pérez

A mi familia, a mis amigos
y a quienes no puedieron verme graduarme.

Índice general

Summary

Keywords: public key cryptosystem, knapsack problem, LLL algorithm, SageMath, and lattice.

What is cryptography?

Cryptography (from Ancient Greek kryptós, «secret», and graphein, «to write») is a discipline that deals with information security in communication through insecure channels. Essentially, it seeks to ensure the confidentiality, integrity, and authenticity of information, enabling efficient and secure communication.

These three terms are probably familiar to any reader, but far from being synonymous, they have clearly differentiated meanings. On one hand, the purpose of confidentiality is to ensure that only authorized individuals or entities can access specific and protected information. On the other hand, integrity seeks to protect information from unauthorized alteration, corruption, or destruction. Finally, authenticity aims to ensure that the information comes from a legitimate source and has not been altered during transmission. In this paper, we will discuss some techniques to ensure confidentiality in communications.

When designing a cryptosystem, it is necessary to consider certain desirable attributes in a communications system, such as efficiency and resistance to cryptographic attacks. Obviously, this is not a simple task, and it is precisely this search that motivates this work.

Brief history of cryptography

The history of cryptography goes back thousands of years to the earliest civilizations. The oldest known use of cryptography is found in non-standard hieroglyphs carved on Ancient Egyptian monuments more than 4500 years ago.

In addition, the Hebrew used monoalphabetic substitution ciphers, marking the beginning of symmetric cryptography. The classics, both Greeks and Romans, were knowledgeable about such ciphers, which they used in the military field. For example, the former developed the scytale, while the latter used the Caesar cipher. During the Renaissance in Europe, the cryptanalytic techniques that emerged were not as secure as claimed. Even today, this

over-optimism is inherent in cryptography, as it is fundamentally difficult to determine the vulnerability of a system.

It is during World War II when the development of cryptography as a discipline boomed, due to the need to decipher messages transmitted on the battlefield. The Germans created the Enigma machine for this purpose, famous for its decryption difficulty. It was not until 1945 that the British, led by Alan Turing, managed to break the code of this machine. This event laid the foundations for modern computing.

Until 1976, the type of encryption used was symmetric, which implies the use of a unique key, both for encrypting and decrypting messages. However, both parties involved need to know the key, and the transmission of the key could lead to potential security vulnerabilities.

Thus, Diffie and Hellman introduced a different encryption method, giving rise to asymmetric cryptography. This method involves generating two keys, a public key and a private key, so that anyone wishing to send a message uses the public key to encrypt it and the private key to decrypt it.

Asymmetric cryptography is based on computationally complex problems, such as the knapsack problem. This problem is just one of several NP-complete problems used in this context.

Next, we proceed to classify asymmetric key or public key cryptosystems according to their methodology:

- Knapsack-problem-Based
- Lattice-Based
- Code-Based
- Elliptic-Curve-Isogeny-Based
- Hash-Based
- Multivariate-Quadratic-Equations

Nevertheless, the appearance in 1994 of a quantum algorithm capable of factoring numbers exponentially faster than classical algorithms, generated a new categorization of asymmetric cryptosystems, dividing them into those resistant to quantum attacks and those vulnerable to these attacks.

Finally, in this work we will focus on analyzing public key cryptosystems, focusing on those based on the knapsack problem, which constitute the title of this project.

Main objectives of the work

The aim of this paper is to provide an introduction to the field of cryptography, starting from the necessary mathematical formalism and ending in a practical implementation of the cryptosystems explained.

We will analyze the theory behind each cryptosystem. In this way, we will see the properties of each one of them, and we will study the algebraic concepts necessary for their understanding. In particular, we will deal with concepts such as modular reduction, orthogonalization and lattices.

On the other hand, several of the cryptosystems explained have been implemented. For this purpose, Python has been used as programming language, together with the support of SageMath libraries. Likewise, we will perform an efficiency analysis of these programs and a comparison that will support the theory studied.

All these objectives have been achieved.

Description and structure of the work

First, chapter 1 will provide the reader with a basic understanding of modern cryptography and how it is applied in the protection of information in computer systems. We also delve into certain cryptosystems, such as AES and RSA, without forgetting historical concepts like the Diffie-Hellman key exchange. Thus, the reader will be able to easily understand everything explained in later chapters.

In chapter 2, we will begin by understanding the seven millennium problems, with the aim of focusing on the P vs NP problem. This development will be aimed at justifying the definition of the NP-Complete problem. Thanks to this definition, we can formally describe the knapsack problem, the basis of the cryptosystems that we will study later.

The chapter 3 aspires to become a guide for those who seek to understand the Merkle-Hellman cryptosystem. We will distinguish different methods, and after studying them, we will provide examples that will help to understand them. Additionally, we will explore the applications of this cryptographic tool, although its weaknesses will be revealed in the next chapter.

The chapter 4 is an analysis of the cryptanalysis proposed by Adi Shamir against the cryptosystem presented in the previous chapter. After an extensive description of the theory behind the attack, we will reach the main result that establishes an upper bound for the failure probability, thus breaking the Merkle-Hellman cryptographic system.

In chapter 5, we will study another type of cryptographic attacks: low-density attacks. Among these, we will highlight the Lagarias-Odlyzko attack and the Coster attack. Both will consist of applying a reduction of the knapsack problem to the search for the shortest vector of a lattice. Therefore, it will be necessary to explain the LLL algorithm, which will

allow us to find such a vector. The chapter will conclude with an experimental analysis of the obtained data.

Finally, in chapter 6, we will describe the Chor-Rivest cryptosystem, the last one in this work. Its key generation method is entirely different from the studied cryptosystems, since it is based on discrete logarithm calculations in finite fields.

Main sources consulted

Among all the bibliographic sources consulted, the following stand out:

[26] has been fundamental for the introduction and the first two chapters.

[18] is the main article that has been used for the writing of chapter 3.

Chapter 4 is taken almost exclusively from the original article by Shamir, [22].

[13], [15] y [10], have been used for each of the sections of the chapter 5.

Finally, chapter 6 has been written thanks to the references [5] y [9].

Introducción

Palabras clave: criptosistema de clave pública, problema de la mochila, algoritmo LLL, SageMath y retículo.

¿Qué es la criptografía?

La criptografía (del griego kryptós, «secreto», y graphé, «grafo», literalmente «escritura secreta») es una disciplina que se ocupa de la seguridad de la información en la comunicación a través de canales inseguros. En esencia, busca garantizar la confidencialidad, la integridad y la autenticidad de la información, permitiendo una comunicación eficiente y segura.

Probablemente, estos tres términos resulten familiares ante cualquier lector, pero lejos de ser sinónimos, tienen significados claramente diferenciados. Por un lado, la confidencialidad tiene como propósito garantizar que únicamente individuos o entidades autorizadas puedan acceder a cierta información específica y protegida. Por otro lado, la intregridad busca proteger la información ante la alteración, corrupción o destrucción no autorizada. Finalmente, la autenticidad tiene como objetivo asegurar que la información provenga de una fuente legítima y que no haya sido alterada durante la transmisión. En este trabajo trataremos algunas técnicas para garantizar la confidencialidad en las comunicaciones.

A la hora de diseñar un criptosistema, es necesario tener en cuenta ciertos atributos deseables en un sistema de comunicaciones, tales como la eficiencia y la resistencia ante ataques criptográficos. Evidentemente, esto no es una tarea sencilla, y es justamente esta búsqueda la que motiva este trabajo.

Breve historia de la criptografía

La historia de la criptografía se remonta miles de años atrás a las primeras civilizaciones. El uso más antiguo conocido de la criptografía se halla en jeroglíficos no estándares tallados en monumentos del Antiguo Egipto de hace más de 4500 años.

También, los eruditos hebreos utilizaron cifrados por sustitución monoalfabéticos, marcando así el inicio de la criptografía simétrica. Los clásicos, tanto griegos como romanos, eran conocedores de cifrados de este tipo que utilizaban en el terreno militar. Por ejemplo, los primeros elaboraron la escítala, mientras que los segundos utilizaron el cifrado Cesar.

Durante el Renacimiento en Europa, las técnicas criptoanalíticas que surgieron no eran tan seguras como afirmaban. Incluso a día de hoy, este sobre optimismo es algo inherente a la criptografía, ya que es fundamentalmente difícil saber cómo de vulnerable es un sistema.

Es a partir de la Segunda Guerra Mundial, cuando se produce el auge en el desarrollo de la criptografía como disciplina, ante la necesidad de descifrar mensajes transmitidos en el campo de batalla. Los alemanes crearon la maquina Enigma con este fin, famosa por su dificultad de descifrado. No fue hasta 1945 que los británicos, liderados por Alan Turing, consiguieron romper el código de esta máquina. Este acontecimiento asentó las bases de la computación moderna.

Hasta el año 1976, el tipo de cifrado utilizado era el simétrico, el cual implica el uso de una clave única tanto para cifrar, como para descifrar el mensaje. Sin embargo, es necesario que ambas partes involucradas conozcan la clave para aplicar este método, y la transmisión de dicha clave puede generar posibles fallos de seguridad.

Así, Diffie y Hellman introducen un método de cifrado distinto, dando origen a la criptografía asimétrica. Este método consiste en generar dos claves, una pública y una privada, de manera que cualquier individuo que desee enviar un mensaje, utilice la clave pública para cifrarlo, y la privada para descifrarlo.

La criptografía asimétrica se fundamenta en problemas computacionalmente muy complejos, como el problema de la mochila. Este problema es simplemente uno de los diversos problemas NP-completos que se emplean en este contexto.

A continuación, procedemos a clasificar los criptosistemas de clave asimétrica o clave pública según su metodología:

- Basados en el problema de la mochila
- Basados en retículos
- Basados en códigos
- Basados en curvas elípticas
- Basados en funciones hash
- Basados en ecuaciones cuadráticas multivariantes

Sin embargo, la aparición en 1994 de un algoritmo cuántico capaz de factorizar números exponencialmente más rápido que los algoritmos clásicos, generó una nueva categorización de los criptosistemas asímetricos, diviendolos en resistentes a ataques cuánticos y vulnerables a estos ataques.

Finalmente, en este trabajo nos centraremos en analizar los criptosistemas de clave pública, enfocándonos en aquellos basados en el problema de la mochila, que constituyen el título de este proyecto.

Principales objetivos del trabajo

El objetivo de este trabajo es ofrecer una introducción al campo de la criptografía, partiendo del formalismo matemático necesario y terminando en una implementación práctica de los criptosistemas explicados.

Analizaremos la teoría detrás de cada criptosistema. De este modo, veremos las propiedades de cada uno de ellos, y estudiaremos los conceptos algebraicos necesarios para su compresión. En particular, abordaremos conceptos como la reducción modular, la ortogonalización y los retículos.

Por otra parte, han sido implementados varios de los criptosistemas explicados. Para ello, se ha utilizado Python como lenguaje de programación, junto al soporte de las librerías de SageMath. Asimismo, realizaremos un análisis de eficiencia de estos programas y una comparativa que respaldará la teoría estudiada.

Todos estos objetivos han sido satisfechos.

Descripción y estructura del trabajo

En primer lugar, el Capítulo 1 proporcionará al lector una comprensión básica de la criptografía moderna y de cómo se aplica en la protección de información de sistemas informáticos. También nos adentramos en ciertos criptosistemas, como AES y RSA, sin olvidar conceptos históricos como el intercambio de claves de Diffie-Hellman. Así, el lector podrá entender sin dificultades todo lo explicado en capítulos posteriores.

En el Capítulo 2, comenzaremos comprendiendo los siete problemas del milenio, con el objetivo de centrarnos en el problema P vs NP. Este desarrollo irá encaminado a justificar la definición de problema NP-Completo. Gracias a esta definición, podremos describir formalmente el problema de la mochila, base de los criptosistemas que veremos más adelante.

El Capítulo 3 aspira a convertirse en una guía para todos aquellos que buscan comprender el criptosistema de Merkle-Hellman. Distinguiremos los distintos métodos, y tras estudiarlos, aportaremos ejemplos que ayuden a su compresión. Asímismo, veremos las aplicaciones de esta herramienta criptográfica, aunque en el próximo capítulo se revelarán sus debilidades.

El Capítulo 4 es un análisis del criptoataque propuesto por Adi Shamir al criptosistema presentado en el capítulo anterior. Tras una extensa descripción de la teoría de su ataque, llegaremos al resultado principal que establecerá una cota superior para la probabilidad de fallo, rompiendo así el sistema criptográfico de Merkle-Hellman.

En el Capítulo 5 estudiaremos otro tipo de ataques criptográficos: los ataques por baja densidad. De entre estos, destacaremos el ataque de Lagarias-Odlyzko y el ataque de Coster. Ambos consistirán en aplicar una reducción del problema de la mochila a la búsqueda del

vector más corto de un retículo. Por tanto, será necesario explicar el algoritmo LLL, que nos permitirá encontrar dicho vector. El capítulo concluirá con un análisis experimental de los datos obtenidos.

Finalmente, en el Capítulo 6 describiremos el criptosistema de Chor-Rivest, el último de este trabajo. Su método de obtención de claves es completamente distinto de los criptosistemas estudiados, ya que se basa en el cálculo de logaritmos discretos en cuerpos finitos.

Principales fuentes consultadas

De entre todas las fuentes bibliográficas consultadas, destacan las siguientes:

[26] ha sido fundamental para realizar la introducción y los dos primeros capítulos.

[18] es el artículo principal que se han usado para la redacción del Capítulo 3.

El Capítulo 4 se ha extraido casi exclusivamente del artículo original de Shamir, [22].

[13], [15] y [10], se han utilizado para cada uno de los apartados del Capítulo 5.

Por último, el Capítulo 6 ha sido redactado gracias a las referencias [5] y [9].

1 Criptografía Básica

En este capítulo establecemos los fundamentos esenciales de la Criptografía. Estos conceptos y herramientas servirán como base para comprender los aspectos más avanzados de la seguridad de la información en capítulos posteriores. Para su elaboración, se han utilizado principalmente [26] y [8].

1.1. Introducción

Desde la antigüedad, la criptografía ha sido utilizada para mantener la privacidad y la confidencialidad en la comunicación de mensajes importantes. Hoy en día, se ha convertido en una herramienta crucial para proteger los datos sensibles en sistemas informáticos y en la comunicación en línea.

En este capítulo, se abordarán algunos conceptos fundamentales de la criptografía moderna, incluyendo los sistemas criptográficos simétricos y asimétricos, técnicas de intercambios de claves y los algoritmos de cifrado y descifrado tales como AES o RSA.

La principal tarea que tiene la criptografía es la trasmisión segura de información. Siguiendo la tradición [26], se suele explicar que un usuario A, denominado Alice, quiere enviar un mensaje secreto a un destinatario B, llamado Bob. En este escenario, también se encuentra Eva, que quiere enterarse del mensaje secreto que envía Alice, y estará atenta al canal de comunicación que utilicen Alice y Bob, para interceptar todos sus mensajes.

Para conseguir esto, Alice tendrá que encriptar su mensaje x con una clave K y enviarle a Bob el resultado $y = enc_K(x)$. Por tanto, luego Bob deberá desencriptar el mensaje encriptado y con su propia clave S, para obtener el mensaje original $x = dec_S(y)$.

Obviamente surgen diversas dudas ante esta situación, tales como: ¿Cómo puede hacerle llegar Alice a Bob la clave necesaria para desencriptar el mensaje? ¿Qué métodos son "mejores" para encriptar el mensaje? O en el peor de los casos, ¿qué pasaría si Eva averigua la clave que están usando?

Volviendo al escenario planteado, está claro que, a priori, nos da igual que Eva consiga el mensaje encriptado, siempre y cuando no sea capaz de desencriptarlo. Por tanto, Alice debería utilizar una función que no sea fácil de desencriptar, ya que su mensaje sería vulnerable. Llamamos a esta función, función unidireccional.

Definición 1.1. [26] Una *función unidireccional* es una función f tal que dado x, $f(x) = y$ es fácil de computar, pero que dado y, sea dificil encontrar un x tal que $f(x) = y$. Si además, Bob tiene algún secreto S, el cual le permite encontrar x a partir de y de manera sencilla, diremos que f es una *función trampilla*.

Ejemplo 1.2. [26] Sea $x = (p, q)$ con p y q primos tal que $p < q$ y $f(x) = p \cdot q$.
Es fácil multiplicar ambos números primos para obtener un valor N, pero es muy dificil, a nivel computacional, obtener los primos p y q a partir del valor N. De hecho, no se conoce ninguna función trampilla para esta f. Ésta es la función unidireccional utilizada en el algoritmo RSA, que se explicará más adelante.

En realidad, que Eva "rompa el sistema", no tiene que significar necesariamente que sea capaz de descifrar el mensaje de manera completa. Podría simplemente limitarse a desencriptar una pequeña parte del mensaje, como por ejemplo conocer el idioma en el que está escrito; o incluso buscar palabras clave, como podrían ser "bomba" o "Mastercard", entre otras posibilidades. Obviamente, debe ser inviable poder recuperar tanto el mensaje x, como la clave de Bob S, a partir del mensaje y.

Por otro lado, otro tipo de ataque sería que Eva consiga ver los valores de distintas $enc_k(x)$ para varios x diferentes. Este tipo de ataque se denomina *chosen plaintext attack* (CPA). Entre los diversos objetivos del ataque, el primero que se nos ocurre es recuperar la clave secreta que se utiliza en el cifrado. Por tanto, una característica deseable es la indistinguibilidad.

Definición 1.3. [26] Definimos *indistinguibilidad* en este contexto como la capacidad de no poder distinguir a partir de una imagen $enc_k(x)$, su preimagen x asociada.

Para entender este concepto, supongamos que Eva envía dos textos planos distintos y recibe uno de ellos desencriptado, y_0, de tal forma que ha sido seleccionado entre los dos textos planos con igual probabilidad. En esta situación, Eva debe determinar cual de los dos textos originales ha sido desencriptado, basándose no solo en la aleatoriedad. Así, la indistinguibilidad es la resistencia a este tipo de ataque, de manera que Eva no pueda saber cúal ha sido el texto original escogido. Procedemos a continuación a la explicación de uno de los algoritmos más populares de toda la criptografía simétrica, el algoritmo AES.

1.2. Advanced Encryption Standard (AES)

El estándar de cifrado avanzado (AES) es un esquema de cifrado por bloques creado en Bélgica pero adoptado como un estándar de cifrado por el gobierno de los Estados Unidos. Tras ser anunciado por el *Instituto Nacional de Estándares y Tecnología* (NIST) en 2001, se estandarizó finalmente en Mayo de 2002. Desde 2006, AES es uno de los algoritmos más populares usados en criptografía simétrica.

1.2.1. Historia de AES

A principios de los años 70, un equipo de International Business Machines desarrolló un criptosistema que se conoció como *Estándar de Cifrado de Datos* (DES). La *National Bureau of Standards* (NBS) de EE.UU lo publicó en el 1977 como estandar para la criptografía del gobierno estadounidense, en particular para documentos sensibles pero no clasificados. Como consecuencia, cualquier sistema de software o hardware con capacidades criptográficas que se ofreciera al gobierno estadounidense, tenía que estar basado en DES.

Dado que las ventas a entidades gubernamentales pueden resultar altamente rentables, cualquier compañía que deseara participar en este mercado, debía emplear DES. Como resultado, su uso se generalizó rápidamente. A lo largo de los años, se desarrollaron diversos ataques contra DES, sobre todo basados en el criptoanálisis diferencial y lineal. En respuesta a esto y ante la preocupación por su reducido espacio de claves, DES se reforzó triplicando su número de "rondas", convirtiéndose así en triple-DES o 3-DES.

Desde el principio, los expertos albergaron sospechas, nunca confirmadas, de que la *National Security Agency* (NSA) podría haber construido una función trampilla para DES que le permitiera descifrar mensajes. Ya en 1981, se advirtió que la agencia NSA era capaz de romper DES utilizando un texto plano probable. Se decía que el principal hardware criptoanalítico implicado consistía en cuatro ordenadores CRAY-1 y que el análisis duraba menos de 24 horas de media.

Finalmente, en 1998 *Electronic Frontier Foundation* (EFF) presentó su US$ 250,000 DES breaker. Así, DES estaba muerto para la mayoría de casos prácticos, aunque el estandar no se retiró hasta 2005.

Previamente, en 1997, la agencia sucesora a NBS, US NIST, abrió una competición para crear AES y reemplazar a DES. Los requisitos eran un cifrado por bloques de 128 bits, y longitudes de clave posibles de 128, 192 y 256 bits. Como era de esperar, las especificaciones eran bastante más precisas que las de su competición de 1973, que condujo a la adopción de DES.

Se presentaron a NIST 15 candidatos, pero se redujeron a una pequeña lista de 5 sistemas para Agosto de 1999. Estos candidatos eran:

1. **MARS**: desarrollado por Don Coppersmith de IBM, uno de los creadores principales de DES.
2. **RC6**: desarrollado por Ron Rivest y 3 colaboradores de RSA Laboratories.
3. **Serpent**: desarrollado por Anderson, Biham y Knudsen.
4. **Twofish**: desarrollado por la empresa Counterpane de Bruce Schneier.
5. **Rijndael**: desarrollado por dos criptógrafos belgas llamados Joan Daemen y Vincent Rijmen.

En Octubre del año 2000, NIST anunció que el ganador era Rijndael. Se esperaba que este sistema fuera seguro durante al menos 30 años.

Por lo general, se alabó al NIST por realizar un procedimiento abierto y bien documentado. Uno de sus requisitos que estableció era demostrar de manera creíble que no existían funciones trampilla ocultas, aliviando así algunas de las preocupaciones que habían surgido durante la estandarización del DES en 1977.

Las características que aseguraron el primer puesto para Rijndael son la seguridad, resistencia ante todos los ataques conocidos y la eficiencia en amplia variedad de plataformas. Además, se caracteriza por una descripción algebraica simple con pocas características inexplicables, lo que hace altamente improbable la existencia de una función trampilla. Hasta el año 2023, no se ha identificado ningún ataque efectivo contra él.

1.2.2. Descripción de AES

AES cifra un mensaje de 128 bits utilizando una clave de 128, 192 o 256 bits, que se distinguen por denominaciones como AES-128. Es un cifrado iterado en el que una secuencia de cuatro operaciones se aplica un número determinado de veces. Consta de 10 rondas cuando la longitud de clave es de 128, 12 rondas con longitud 192 y 14 rondas con longitud 256. Para la totalidad de la explicación, trabajaremos con AES-128.

En cada ronda realiza cuatro operaciones, salvo que en la última ronda se omite una de ellas y en la primera se añade otra. Cada operación convierte una palabra de 128 bits en otra de 128 bits, es decir, se mantiene la longitud. Para describir las operaciones, cada palabra de 128 bits (o estado en AES) es tratada como una matriz $4x4$ donde cada elemento a_{ij} con $i, j = 0, 1, 2, 3$, es un byte de 8 bits de la siguiente forma:

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Las operaciones y etapas de AES se explican y se muestran a continuación:

1. **SUBBYTES**: sustituye cada byte por otro valor.
2. **SHIFTROWS**: permuta los bytes de cada fila.
3. **MIXCOLUMNS**: aplica una transformación lineal a cada columna.
4. **ADDROUNDKEY**: añade una clave a toda la matriz.

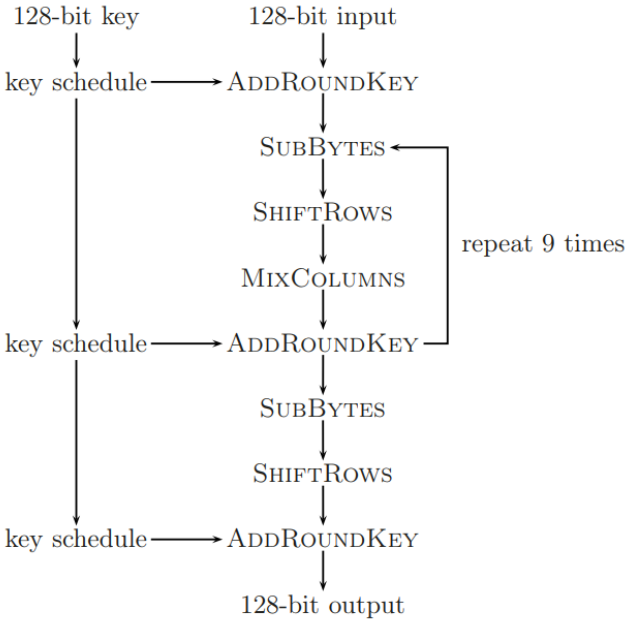

Figura 1.1: Rondas de AES-128 [26].

Seguidamente, vamos a proceder a explicar en más detalle las distintas operaciones que constituyen AES. A medida que avancemos en este documento, también exploraremos otros sistemas de cifrado como RSA o Merkle-Hellman, entre otros. No obstante, de la elegante explicación algebraica que se presenta a continuación, destaca la sorprendente eficacia del álgebra en el campo de la criptografía.

1.2.3. Operaciones de AES

La unidad básica de procesamiento es el byte, formado por 8 bits.

$$a = (a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0) \in \{0,1\}^8$$

Las operaciones básicas de los bytes son la suma y la multiplicación. Las definimos de la siguiente forma:

- **Suma**. Sean $a, b, c \in \{0,1\}^8$, la suma de dos bytes es la suma bit a bit módulo 2:

$$c_i = (a_i + b_i) \bmod 2 \text{ , con } i = 0, ..., 7$$

Ejemplo 1.4. Por ejemplo, sea $a = (1,0,0,1,1,0,1,1)$ y $b = (1,1,0,0,1,1,0,1)$, entonces:

$$c = a + b = (0,1,0,1,0,1,1,0)$$

- **Producto**. Para el producto, consideramos que el byte a está representando el siguiente polinomio:

$$a_7t^7 + a_6t^6 + \ldots + a_1t + a_0 \in \mathbb{F}_2[t]$$

De esta forma, el producto $p = a \cdot b$ se calcula multiplicando ambos polinomios, obteniendo un polinomio de grado 14 como máximo. Aunque ahora tenemos un problema obvio, ya que el resultado obtenido tiene 15 bits pero la operación debe devolver al final solo 8. Para solucionar esto, lo que haremos será reducir módulo un polinomio de grado 8. De hecho, en AES trabajamos en el campo finito $\mathbb{F}_{256}$ definido por el polinomio irreducible:

$$m = t^8 + t^4 + t^3 + t + 1 \in \mathbb{F}_2[t]$$

Por tanto, p mod $m \in \mathbb{F}_2[t]/(m) = \mathbb{F}_{2^8} = \mathbb{F}_{256}$, volviendo de nuevo al grado máximo de 8 bits que nos da el resultado final del producto:

$$c = a_7t^7 + a_6t^6 + a_5t^5 + a_4t^4 + a_3t^3 + a_2t^2 + a_1t + a_0 \in \mathbb{F}_{256}$$
$$\text{con } a_i \in \{0,1\},\ i = 0,...,7$$

Veamos un ejemplo que nos clarifique los conceptos vistos.

Ejemplo 1.5. Sea $a = (1,0,0,1,1,0,1,1)$ y $b = (1,1,0,0,1,1,0,1)$ los mismos valores de antes. Ahora, sin embargo, representan los siguientes polinomios respectivamente:

$$a = t^7 + t^4 + t^3 + t + 1 \in \mathbb{F}_2[t]$$
$$b = t^7 + t^6 + t^3 + t^2 + 1 \in \mathbb{F}_2[t]$$

El producto de estos polinomios es:

$$p = t^{14} + t^{13} + t^{11} + t^{10} + t^8 + t^6 + t^5 + t^3 + t^2 + t + 1 \in \mathbb{F}_2[t]$$

Recordemos que el polinomio irreducible es:

$$m = t^8 + t^4 + t^3 + t + 1 \in \mathbb{F}_2[t]$$

A continuación, calculamos p mod m, esto es, dividimos p entre m y nos quedamos con el resto, obteniendo:

$$p = (t^6 + t^5 + t^3) \cdot m + (t^4 + t^2 + t + 1) \in \mathbb{F}_2[t]$$

Finalmente, fijándonos en el resto, obtenemos que este polinomio es la representación de $c = (0,0,0,1,0,1,1,1)$.

1.2.3.1. SubBytes

La operación SubBytes de AES aplica una transformación a cada byte de la matriz $4x4$. Es por ello que el apartado anterior define las dos operaciones principales que vamos a utilizar, la suma y el producto. Estas operaciones las realizamos a partir de dos bytes, obteniendo otro byte como resultado. El problema es que en SubBytes, solo tenemos un byte de entrada. Para solucionar esto, haremos uso de la inversa.

Como $\mathbb{F}_{256}$ es un cuerpo, todo elemento no nulo $a \in \mathbb{F}_{256}^x$ tiene un inverso $a^{-1} \in \mathbb{F}_{256}^x$. Este elemento inverso se puede calcular mediante el Algoritmo Extendido de Euclides. Definimos esta función en todo $\mathbb{F}_{256}$ simplemente enviando cero a si mismo:

$$\text{inv}(a) = \begin{cases} a^{-1}, \text{ si } a \neq 00 \\ 00, \text{ si } a = 00 \end{cases}$$

donde $00 = (0,0,0,0,0,0,0,0)$. Entendamos lo explicado en el siguiente ejemplo.

Ejemplo 1.6. [26] Tomando los valores de los ejemplos vistos previamente, el Algoritmo Extendido de Euclides nos da la siguiente igualdad:

$$(t^7 + t^3) \cdot a + (t^6 + t^3 + t^2 + t + 1) \cdot m = 1, \text{ en } \mathbb{F}_2[t]$$

De hecho, como gcd(a, m) = 1, entonces:

$$\text{inv}(a) = (1,0,0,0,1,0,0,0) \text{ en } \mathbb{F}_{256}.$$

AES utiliza una estructura algebraica similar, aunque diferente, sobre los bytes, llamada el anillo $R = \mathbb{F}_2[t]/(t^8 + 1)$. Esta estructura no es un cuerpo, ya que $t^8 + 1 = (t + 1)^8$ no es irreducible en $\mathbb{F}_2[t]$. Por tanto, ahora un byte a representa al elemento:

$$a_7t^7 + a_6t^6 + a_5t^5 + a_4t^4 + a_3t^3 + a_2t^2 + a_1t + a_0 \text{ en } R$$

La suma, nuevamente, es la suma de bits módulo dos (o XOR), por lo que es válido en R. En la multiplicación, también se lleva a cabo el producto de polinomios, para luego reducir su grado tomando módulo con un polinomio de grado 8, solo que en este caso, este polinomio de reducción es $m = t^8 + 1$. Esta reducción es particularmente sencilla ya que corresponde a dividir la parte superior e inferior del polinomio producto p con tamaño 16 bits, en dos partes que denominaremos c_1 y c_0 respectivamente, de tamaño 8 bits cada una, para luego sumarlas y obtener el resultado final. Veamos esta idea en un ejemplo que nos resulte claro.

Ejemplo 1.7. Tomando $a = (1,0,0,1,1,0,1,1)$ y $b = (1,1,0,0,1,1,0,1)$ los mismos valores previos, realizamos la multiplicación de los polinomios y obtenemos el mismo resultado anterior:

$$p = t^{14} + t^{13} + t^{11} + t^{10} + t^8 + t^6 + t^5 + t^3 + t^2 + t + 1$$

Recordemos ahora que el polinomio irreducible es:

$$m = t^8 + 1$$

Por tanto, aplicando ahora reducción de p módulo m, obtendríamos lo siguiente:

$$p = (t^6 + t^5 + t^3 + t^2 + t + 1) \cdot m + t$$

Por tanto, el resultado final es $c = (0, 0, 0, 0, 0, 0, 1, 0)$, ya que debemos centrarnos solo en el resto. No obstante, lo particular de este polinomio irreducible es que es más fácil su cálculo, ya que podemos dividir el polinimio p en dos partes, que hemos denominado c_1 y c_0. Esta idea se expresa en la siguiente igualdad:

$$p = c_1 \cdot t^8 + c_0$$

Así, para calcular cada c_i, debemos dividir del polinomio p con tamaño 16 bits, en dos mitades con 8 bits cada una, de tal forma que obtenemos:

$$c_1 = (0, 1, 1, 0, 1, 1, 0, 1)$$
$$c_0 = (0, 1, 1, 0, 1, 1, 1, 1)$$

Para finalizar, solo debemos sumar ambas c_i usando la suma en R y obtendremos el resultado final $c = (0, 0, 0, 0, 0, 0, 1, 0)$, que coincide con el que habíamos obtenido.

Tras haber explicado todo esto, cabe mencionar que en AES, realmente sólo se realiza el producto en R por el polinomio fijo t_1, y solo se suma el polinomio t_0. Estos son:

$$t_1 = (0, 0, 0, 1, 1, 1, 1, 1) = t^4 + t^3 + t^2 + t + 1$$
$$t_0 = (0, 1, 1, 0, 0, 0, 1, 1) = t^6 + t^5 + t + 1$$

Al ser t_1 invertible módulo $t^8 + 1$, la multiplicación por t_1 corresponde a una transformación lineal invertible sobre $\mathbb{F}_2$. Así, para un byte a, debemos aplicar:

$$c = t_1 \cdot b + t_0$$

donde $b = \text{inv}(a)$, visto previamente. Esto también puede ser descrito por una transformación lineal de la siguiente forma:

$$\begin{bmatrix} c_7 \\ c_6 \\ c_5 \\ c_4 \\ c_3 \\ c_2 \\ c_1 \\ c_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Finalmente como resumen, SubBytes consiste en aplicar a cada byte a del bloque los siguientes pasos:

$$a \leftarrow \text{inv}(a) \text{ , en } \mathbb{F}_{256}$$
$$a \leftarrow t_1 \cdot a \text{ , en } R$$
$$a \leftarrow a + t_0 \text{ , en } R$$

SubBytes es la única operación no lineal de AES. Como curiosidad, a veces es llamada "S-box", en analogía a las funciones no lineales de DES.

1.2.3.2. ShiftRows

La operación ShiftRows de AES desplaza cada una de las cuatro filas cíclicamente hacia la izquierda en 0, 1, 2 y 3 posiciones, respectivamente, ya que recordemos que estamos trabajando con AES-128.

Ejemplo 1.8. De manera genérica, aplicando ShiftRows a nuestra matriz inicial, obtendremos:

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \rightarrow \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{11} & a_{12} & a_{13} & a_{10} \\ a_{22} & a_{23} & a_{20} & a_{21} \\ a_{33} & a_{30} & a_{31} & a_{32} \end{bmatrix}$$

1.2.3.3. MixColumns

Como ya hemos mencionado, la operación MixColumns de AES aplica una transformación lineal a cada columna de la matriz $4x4$ de bytes. Para ello, vamos a utilizar una estructura algebraica similar a la explicada en SubBytes. Consideramos cada vector $a = (a_3, a_2, a_1, a_0)$ de 4 bytes como un polinomio de grado máximo 3:

$$a_3 \cdot t^3 + a_2 \cdot t^2 + a_1 \cdot t + a_0 \in \mathbb{F}_{256}[t]$$

Donde la suma corresponde a la suma de bits módulo dos (o XOR) y el producto corresponde al ya explicado, con la diferencia de que el polinomio irreducible en este caso es $t^4 + 1 \in \mathbb{F}_{256}[t]$. Entonces, ahora trabajamos en el anillo $R = \mathbb{F}_{256}[t]/(t^4 + 1)$, con 256^4 elementos. Al igual que antes, $t^4 + 1 = (t + 1)^4$ no es irreducible en $\mathbb{F}_{256}[t]$, por lo que R no es un cuerpo. Además, la reducción módulo el polinomio irreducible es también particularmente sencilla, ya que podemos dividir el polinimio producto p en dos partes, que denominamos c_0, $c_1 \in \mathbb{F}_{256}[t]$, con grado máximo 3, como se muestra en la siguiente igualdad:

$$p = c_1 \cdot t^4 + c_0$$

De esta forma, para calcular el producto solo debemos sumar ambas c_i usando la suma en R para obtener el resultado final.

Sin embargo podemos verlo de otra forma, ya que el producto puede ser descrito como un vector de 4 bytes $b = (b_3, b_2, b_1, b_0)$, dado por el siguiente producto matriz por vector:

$$\begin{bmatrix} b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} = \begin{bmatrix} 02 & 01 & 01 & 03 \\ 03 & 02 & 01 & 01 \\ 01 & 03 & 02 & 01 \\ 01 & 01 & 03 & 02 \end{bmatrix} \cdot \begin{bmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix}$$

Donde las operaciones entre bytes individuales son las de $\mathbb{F}_{256} = \mathbb{F}_2[t]/(t^4 + 1)$, es decir, las explicadas arriba. Veamos a continuación un ejemplo de MixColumns.

Ejemplo 1.9. [26] Sea $a = (\text{A0}, 80, 01, 02)$ una columna de la matriz de bytes, aplicamos MixColumns de la siguiente forma:

$$\begin{bmatrix} b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} = \begin{bmatrix} 02 & 01 & 01 & 03 \\ 03 & 02 & 01 & 01 \\ 01 & 03 & 02 & 01 \\ 01 & 01 & 03 & 02 \end{bmatrix} \cdot \begin{bmatrix} \text{A0} \\ 80 \\ 01 \\ 02 \end{bmatrix}$$

Así,

$$\begin{aligned} b_3 &= 02 \cdot \text{A0} + 01 \cdot 80 + 01 \cdot 01 + 03 \cdot 02 \\ &= t \cdot (t^7 + t^5) + 1 \cdot t^7 + 1 \cdot 1 + (t+1) \cdot t \\ &= t^8 + t^7 + t^6 + t^2 + t + 1 \\ &= t^7 + t^6 + t^4 + t^3 + t^2 \\ &= (1, 1, 0, 1, 1, 1, 0, 0) \\ &= \text{FC en } \mathbb{F}_{256} \end{aligned}$$

Donde hemos tenido en cuenta que $t^8 = t^4 + t^3 + t + 1$ en $\mathbb{F}_{256}$. Analogamente,

$$\begin{aligned} b_2 &= 03 \cdot \text{A0} + 02 \cdot 80 + 01 \cdot 01 + 01 \cdot 02 \\ &= (t+1) \cdot (t^7 + t^5) + t \cdot t^7 + 1 \cdot 1 + 1 \cdot t \\ &= t^7 + t^6 + t^5 + t + 1 \\ &= (1, 1, 1, 0, 0, 0, 1, 1) \\ &= \text{E3 en } \mathbb{F}_{256} \end{aligned}$$

$$\begin{aligned} b_1 &= 01 \cdot \text{A0} + 03 \cdot 80 + 02 \cdot 01 + 01 \cdot 02 \\ &= 1 \cdot (t^7 + t^5) + (t+1) \cdot t^7 + t \cdot 1 + 1 \cdot t \\ &= t^5 + t^4 + t^3 + t + 1 \\ &= (0, 0, 1, 1, 1, 0, 1, 1) \\ &= \text{3B en } \mathbb{F}_{256} \end{aligned}$$

$$
\begin{aligned}
b_0 &= 01 \cdot A0 + 01 \cdot 80 + 03 \cdot 01 + 02 \cdot 02 \\
&= 1 \cdot (t^7 + t^5) + 1 \cdot t^7 + (t+1) \cdot 1 + t \cdot t \\
&= t^5 + t^2 + t + 1 \\
&= (0,0,1,0,0,1,1,1) \\
&= 27 \text{ en } \mathbb{F}_{256}
\end{aligned}
$$

Finalmente,

$$
b = \begin{bmatrix} b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} = \begin{bmatrix} FC \\ E3 \\ 3B \\ 27 \end{bmatrix}
$$

Obteniendo así el resultado final de aplicar MixColumns al vector columna a.

1.2.3.4. AddRoundKey

La operación AddRoundKey de AES suma bit a bit un bloque de 128 bits con una clave de ronda de su mismo tamaño. Como ya hemos explicado, AES permite claves de 128, 192 o 256 bits. Estas claves corresponden a l_k palabras de 32 bits, con $l_k = 4, 6$ o 8. Además, el número de rondas tras la inical, denominado l_r, se muestra en la siguiente tabla:

key length		
in bits	in ℓ_k words	ℓ_r rounds
128	4	10
192	6	12
256	8	14

Figura 1.2: Número de palabras l_k y de rondas l_r de AES [26].

En realidad, cada palabra tiene la forma de un vector columna, simplemente l_k indica cuantas filas componen a ese vector. En esta operación buscamos en cada ronda generar una clave de ronda, que será un vector con tantas componentes como rondas l_k haya, más una para la ronda inicial. Por tanto, se requiere un total de $l_k \cdot (l_r + 1)$ claves de ronda.

Ejemplo 1.10. Por ejemplo, para una llave de 256 bits, los mensajes consisten en 8 palabras de 4 bytes cada una, equivalentemente, 8 palabras de 32 bits cada una. Esto lo sabemos ya que a partir de la figura 1.2, la clave está formada por $l_k = 8$ palabras del tipo $K_0, ..., K_7$. Además, sabemos que el número de rondas es $l_r = 14$, por lo que necesitaremos una clave extendida de $l_k(l_r + 1) = 8 \cdot (14 + 1) = 120$ palabras.

Procedemos a continuación a explicar esta operación para AES-128, es decir, tomando $l_k = 4$ y $l_r = 10$. La clave secreta K está formada por las cuatro primeras palabras de 32 bits E_0, E_1, E_2 y E_3 de la clave extendida $E_0, ..., E_{4(l_r+1)-1}$, que consta de $4(l_r + 1)$ palabras de 32 bits. El resto se generarán a partir de las que ya tenemos. Así, nuestras claves de ronda consistirán en bloques de 4 palabras consecutivas de la clave extendida. La forma de calcular el resto de palabras es la siguiente:

$$E_i = E_{i-1} + E_{i-4} \text{ , con } i \geq 4$$

Esto no es más que la suma en $\mathbb{Z}_2^{32}$, en otras palabras, la suma de bits módulo dos o XOR, vista anteriormente.

No obstante, si i es múltiplo de 4, se aplica primero una transformación al elemento E_{i-1} tal que sus elementos (a_0, a_1, a_2, a_3) se desplazan hacia la derecha cíclicamente, obteniendo (a_3, a_0, a_1, a_2). Es más, si tomamos cada palabra como un elemento a de $S = \mathbb{F}_{256}[t]/(t^4 + 1)$, consiste simplemente en multiplicar por t^3 en S. Despues, se aplica SubBytes a cada byte individualmente, y se añade una constante $c_{i/4}$, definida de esta forma:

$$c_j = (0, 0, 0, t^{j-1}) = t^{j-1} \text{ en } S$$

Decimos que es una constante porque, tomandola como un elemento de S, tiene la forma $a_3t^3 + a_2t^2 + a_1t + a_0$, con $a_3 = a_2 = a_1 = 0$.

Por último, en el caso en que trabajemos con AES-256 y por tanto $l_k = 8$ y $l_r = 14$, se aplica otra transformación. Consiste en que E_{i-1}, se reemplaza por SubBytes(E_{i-1}), si i mod 8 = 4. Poniendo toda esta información en común, obtenemos este algoritmo explicativo:

Algorithm 1 Algoritmo de clave extendida de AES

Input: Clave $K_0, ..., K_{l_k-1}$ formada por l_k palabras, donde cada K_i tiene 4 bytes.
Output: Clave extendida $E_0, ..., E_{l_k(l_r+1)-1}$ formada por $l_k(l_r + 1)$ palabras, donde cada E_i tiene 4 bytes.

1: **for** i from 0 to $l_k - 1$ **do**
2: $E_i = K_i$
3: **end for**
4: **for** i from l_k to $l_k \cdot (l_r + 1)$ **do**
5: $L = E_{i-1}$
6: **if** l_k divides i **then**
7: $c = (0, 0, 0, t^{(i/l_k)-1})$
8: L = SubBytes$(s^3 \cdot L) + c$
9: **end if**
10: **if** $l_k == 8$ and i mod 8 == 4 **then**
11: L = SubBytes(L)
12: **end if**
13: $E_i = E_{i-l_k} + L$
14: **end for**

Una vez hemos descrito todas las operaciones de AES para una ronda, solo nos hace falta repetirlas en el orden y el número de veces que indica la figura 1.1 del principio. Aunque debemos tener en cuenta que dependiendo de qué AES estemos utilizando, el número de rondas será distinto. Con esto, damos por concluida nuestra descripción general del cifrado de AES.

1.2.4. Comentarios sobre AES

Primeramente, debemos saber que AES evolucionó a partir de los cifrados por bloques desarrollados por sus diseñadores, previos a Rijndael. Tales como "SHARK", desarrollado por Vincent Rijmen; y "Square", desarrollado por Joan Daemen. Así, su filosofía de diseño perseguía un alto nivel de rendimiento y resistencia contra el criptoanálisis lineal y diferencial.

Como ejemplos, destacamos que para SubBytes, fué sugerida la idea de utilizar la operación de inversión, y que el módulo m es el primero de los 30 polinomios irreducibles de la Tabla C del capítulo 10 de "Finite Fields" de Rudolf Lidl & Harald Niederreiter [16]. MixColumns además se basa en matrices en las que cada submatriz cuadrada es no singular, una noción explicada en el teorema 8 del capítulo 11 de "The theory of error-correcting codes" de MacWilliams & Sloane [17].

Asimismo, la omisión de MixColumns en la última ronda es bastante común, ya que no disminuye la seguridad (porque los bits del texto cifrado sólo se permutan de una forma públicamente conocida), pero permite el descifrado con una estructura similar.

En cuanto a su rendimiento, tal y como se exige en la competición de AES, el algoritmo es rápido en una gran variedad de plataformas. Las implementaciones de software pueden alcanzar más de 12 GB/seg. Aun así, si tenemos como objetivo realizar una implementación de software, debemos saber que generalmente es ventajoso sustituir los cálculos por consultas de tablas. Utilizando una tabla de 4 KB, una ronda de AES puede ejecutarse con 16 búsquedas en la tabla y 16 XOR de 32 bits.

Por otro lado, en cuanto a su seguridad, sí que es cierto que algunos criptógrafos muestran preocupación, ya que sienten que el margen entre el número de rondas especificado en el cifrador y los mejores ataques conocidos es muy pequeño. Además, se especula con ataques teóricos, como por ejemplo el denominado "Ataque XSL", que teóricamente muestra una potencial debilidad en el algoritmo AES. Sin embargo, tanto los expertos como las instituciones de normalización más relevantes consideran que AES es seguro.

1.3. Criptografía simétrica vs asimétrica

Hasta ahora, solo hemos trabajado con AES, un criptosistema de tipo simétrico. Este tipo de criptosistemas se caracterizan porque la misma clave que se utiliza para cifrar el mensaje, es la que se usa para descifrarlo. Al igual que AES, todos los cifrados previos a 1970 eran de este tipo.

Sin embargo, Diffie & Hellman [8] hicieron una propuesta revolucionaria en 1976 publicando su libro "New directions in cryptography", en el que presentaron un algoritmo que demostró que la criptografía asimétrica era posible. En él, proponen un nuevo método de comunicación donde cada usuario posee dos claves, una privada y una pública. Digamos que Alice quiere enviar un mensaje a Bob. Entonces Alice debe utilizar la clave pública de Bob para encriptar el mensaje que le mandará. Gracias a la clave secreta, Bob puede fácilmente desencriptar el mensaje de Alice, pero sin ella, nadie debería poder.

Con la llegada de este nuevo tipo de criptosistemas, se comenzó a estudiar si estos eran mejores o peores que los ya conocidos criptosistemas simétricos. Pero como en todo, se encontraron ventajas y desventajas en cada uno de ellos.

Por un lado, la ventaja de los criptosistemas asimétricos radica en que no necesitan un cambio de clave. Por otro, los criptosistemas simétricos tienen ventaja de velocidad, ya que como hemos mencionado, AES puede alcanzar 12 GB/seg. Otra ventaja de estos últimos es la autenticación, puesto que en los criptosistemas asimétricos debemos tomar medidas para garantizar esta característica.

Actualmente, la realidad es que no existe una rivalidad entre criptografía "Simétrica vs Asimétrica". Por lo general, se utiliza un método asimétrico para compartir las claves necesarias y así, posteriormente, poder aplicar un criptosistema simétrico permitiendo comunicación de mayor rendimiento.

Veamos a continuación dos ejemplos de criptosistemas asimétricos para conocerlos un poco más a fondo, comenzando por el famoso criptosistema RSA.

1.4. Criptosistema RSA

Basándose en el primer modelo abstracto de criptosistema de clave pública sugerido por Diffie & Hellman en 1976, Rivest, Shamir & Adleman crearon su propio criptosistema asimétrico en 1977 llamado "Rivest-Shamir-Adleman cryptosystem", o más conocido como RSA.

Continuemos con la notación de nombres utilizada hasta ahora. En nuestro escenario, Alice quiere enviar un mensaje a Bob, que sólo él sea capaz de leerlo. Para ello, Bob genera dos claves: una clave pública pk y una clave privada sk. Cualquiera puede conocer la clave pública pk. Puede estar expuesta en una base de datos o en internet, mismamente. Sin embargo, la clave privada sk sí que debe ser guardada por Bob en secreto. De esta forma,

Alice usará la clave pública pk para encriptar su mensaje y Bob usará su clave privada sk para descifrarla.

En los criptosistemas simétricos, el cifrado y descrifrado se realiza esencialmente con la misma clave, pero aquí, pk y sk son claves completamente distintas. De hecho, deben cumplir algunas propiedades como que sk no debe ser computable facilmente a partir de pk; y deben verificar que $dec_{sk}(enc_{pk}(x)) = dec_{pk}(enc_{sk}(x)) = x$, siendo x el mensaje original.

El mensaje puede ser una imagen, un texto, un video, o incluso un programa. Pero vamos a suponer que ya ha sido convertido a un string binario (posiblemente muy largo), que será nuestra forma de comunicación estándar. Esta conversión dependerá del tipo de dato que se trate, como por ejemplo, la manera más común de enviar un texto es haciendo uso de la codificación ASCII o ASCII extendida, en letras de 7 u 8 bits, respectivamente. Aunque en la práctica, RSA se utiliza para enviar una clave secreta y así establecer un criptosistema simétrico; o para garantizar autenticidad mediante firmas digitales.

En nuestro escenario, estamos suponiendo que Alice quiere enviarle un mensaje a Bob. Para ello, Alice debe tener en cuenta el valor n, denominado *parámetro de seguridad*, ya que debe dividir el mensaje x en bloques de tamaño $n-1$ bits y enviar cada bloque por separado. Por tanto, vamos a explicar como se envía un bloque de $n-1$ bits usando el criptosistema de RSA.

En primer lugar, Bob debe elegir dos valores primos p y q aleatorios, con $\frac{n}{2}$ bits cada uno. Así, su producto $N = p \cdot q$ tendrá n bits. Además, Bob debe elegir un número entero e aleatorio, tal que $1 \leq e < N$ y $\gcd(e, (p-1)(q-1)) = 1$, es decir, e y el producto $(p-1) \cdot (q-1)$ sean coprimos. Así, la clave pública de Bob es $pk = (N, e)$.

A continuación, a partir del mensaje x, Alice envía a Bob el mensaje encriptado $y = x^e$ en $\mathbb{Z}_n$, esto es, el resto de dividir x^e entre N. La magia es que Bob ahora puede recuperar el mensaje de Alice x con la ayuda de su información secreta relativa a (p, q). El algoritmo es el siguiente:

Algorithm 2 Algoritmo de generación de claves de RSA

Input: Parámetro de seguridad n.
Output: Clave privada sk y clave pública pk.

1: Elegir los valores primos p, q tal que $2^{(n-1)/2} < p$ y $q < 2^{n/2}$.
2: Calcular $N = p \cdot q$ con n bits y $L = (p-1) \cdot (q-1) = \phi(p) \cdot \phi(q) = \phi(N)$, es decir, la función de Euler aplicada a N.
3: Escoger e de manera uniformemente aleatoria del conjunto $\{2, ..., L-2\}$, tal que sea coprimo con L. Así $e \in \mathbb{Z}_L$.
4: Calcular el inverso d de e en $\mathbb{Z}_L$. Así $d \in \mathbb{Z}_L$.
5: Revelar la clave pública $pk = (N, e)$ y esconder la clave privada $sk = (N, d)$.

Algorithm 3 Algoritmo de cifrado de RSA

Input: Mensaje $x \in \mathbb{Z}_N$ y clave pública $pk = (N, e)$.
Output: Mensaje cifrado $y = enc_{pk}(x) \in \mathbb{Z}_N$.

1: Aplicar la transformación $y = x^e$ en $\mathbb{Z}_N$.
2: Devolver el mensaje cifrado $y = enc_{pk}(x)$.

Algorithm 4 Algoritmo de descifrado de RSA

Input: Mensaje cifrado $y \in \mathbb{Z}_N$ y clave privada $sk = (N, d)$.
Output: Mensaje original $x = dec_{sk}(y) \in \mathbb{Z}_N$.

1: Aplicar la transformación $x = y^d$ en $\mathbb{Z}_N$.
2: Devolver el mensaje original $x = dec_{sk}(y)$.

Ejemplo 1.11. Veamos un ejemplo tomando $n = 6$. Si en este caso comprobamos las condiciones de manera literal, debemos escoger números primos entre 6 y 7, pero al ser valores tan pequeños, tomaremos la libertad de ser un poco más liberales y escogeremos $p = 7$ y $q = 11$. Así, $N = 77$ y $L = 60$. Tomando $e = 17$ y aplicando el Algoritmo Extendido de Euclides, obtenemos que $d = e^{-1} = 53$ en $\mathbb{Z}_{60}$. De esta forma, Bob ya puede publicar su clave pública $pk = (77, 17)$ y guardar su clave privada $sk = (77, 53)$.

Posteriormente, Alice quiere mandarle un mensaje a Bob, pongamos $x = 10 = (00001010)$. Entonces Alice debe cifrar el mensaje $y = x^e = 10^{17}$ en $\mathbb{Z}_{77}$. La forma intuitiva para calcular esto sería computar 10^{17} y aplicarle el módulo 77. Pero esto tiene un problema, y es que para valores reales de n, x^e puede superar el número de partículas en el universo. Sin embargo, hay una forma más sencilla de calcularlo. La idea es descomponer el valor en sus potencias de dos y aplicar módulo tras cada operación para no dejar que el valor se dispare. Veamoslo:

Ya que $(17)_{10} = (10001)_2 \Rightarrow 10^{17} = 10^{2^0+2^4} = 10 \cdot 10^{16}$
Calculamos así las potencias de 10 mod 77:

$10 \bmod 77 = 10$

$10^2 \bmod 77 = 10 \cdot 10 \bmod 77 = 23$

$10^4 \bmod 77 = 10^2 \cdot 10^2 \bmod 77 = 23^2 \bmod 77 = 67$

$10^8 \bmod 77 = 10^4 \cdot 10^4 \bmod 77 = 67^2 \bmod 77 = 23$

$10^{16} \bmod 77 = 10^8 \cdot 10^8 \bmod 77 = 23^2 \bmod 77 = 67$

Por tanto,

$10^{17} \bmod 77 = 10 \cdot 10^{16} \bmod 77 = 10 \cdot 67 \bmod 77 = 54$

Ahora, Alice ya puede enviar su mensaje cifrado $y = enc_{pk}(10) = 54$ a Bob. Por tanto, Bob debe descifrar el mensaje recibido, y para ello, debe usar de manera análoga la descomposición binaria de 53 (que es el valor de su clave secereta sk) para descifrar el mensaje en $\mathbb{Z}_{77}$. De esta forma, $dec_{sk}(54) = x = 54^{53} = 10$ en $\mathbb{Z}_{77}$. De hecho, ese es el mensaje original que Alice quería hacerle llegar a Bob.

Observación 1.12. Tal y como ya hemos mencionado antes, debemos dejar claro que los valores tomados en este ejemplo son simplemente para esclarecer las ideas del método propuesto, ya que con objetivos prácticos reales, es recomendable que el valor del parámetro de seguridad sea mayor a $n = 3000$.

1.5. Intercambio de claves Diffie-Hellman

Para los criptosistemas simétricos como AES, se debe elegir una clave secreta y compartirla entre todos las partes autorizadas. En la práctica, este acuerdo plantea una gran dificultad en este tipo de criptosistemas. Para ilustrar esto, comenzaremos esta sección con un ejemplo histórico.

Durante la Primera Guerra Mundial, los alemanes quedaron aislados de sus embajadas en el extranjero. Surgieron las sospechas de que sus antiguos códigos de comunicación estaban rotos y enviaron un código nuevo a México y a Washington en un submarino. Sin embargo, esta clave no pudo ser entregada con éxito a México. A pesar de esto, un mensaje secreto de gran importancia llegó sin problemas a Washington utilizando el nuevo código, aunque, posteriormente tuvo que ser retransmitido a México en el código antiguo, el cual fue descifrado. La implementación en su momento de un método público y seguro para el intercambio de claves secretas podría haber evitado esta situación.

Por tanto, la pregunta es la siguiente: ¿Pueden dos usuarios ponerse de acuerdo sobre una clave secreta compartida (para un criptosistema simétrico) mientras se ven obligados a utilizar un canal inseguro? La sorprendente respuesta es que sí es posible. Obviamente, ninguno puede enviar directamente la clave secreta ya que el canal es inseguro. En lugar de eso, cada usuario debe enviar una versión enmascarada de la clave, a partir de la cual el otro usuario pueda obtenerla, pero un espía no pueda.

Diffie y Hellman proponen el uso de un número primo p, y a partir de él, trabajar con valores que no sean divisibles por p y las propiedades de su producto. Estos valores forman el grupo $\mathbb{Z}_p^x$ de unidades módulo p. En total hay $p-1$ elementos y el producto de dos de ellos tampoco es divisible por p. Además, todos los elementos tienen inverso multiplicativo, que se puede calcular haciendo uso del Algoritmo Extendido de Euclides. Asimismo, este grupo $G = \mathbb{Z}_p^x$ es *cíclico*, esto es, que existe un elemento $g \in G$, llamado *generador*, cuyas potencias generan todos los elementos de $G = \{g^0 = 1, g, g^2, ..., g^{p-2}\} = \langle g \rangle$. El *orden* del grupo, es decir, el número de elementos de G, lo denotaremos como d. Aparece ahora un parámetro de seguridad global n, que indica el tamaño de d en bits. Así, los elementos de G pueden representarse por cadenas de n bits.

Un requisito importante es que la representación de G pueda ser expresada utilizando una cantidad de bits que crezca de manera polinómica respecto al valor n. Otro requisito, es que las operaciones del grupo puedan transformarse en las representaciones de los elementos de G, con una cantidad de operaciones que aumente de manera polinómica respecto a n. En general, estos dos requisitos se cumplen de manera clara.

Algunos ejemplos de G son los siguientes:

- El grupo multiplicativo $G = \mathbb{Z}_p^x$ de unidades módulo p.
- El grupo multiplicativo $G = \mathbb{F}_q^x$ de un cuerpo finito $\mathbb{F}_q$.
- Grupos cíclicos de curvas elípticas sobre cuerpos finitos.

Ejemplo 1.13. [26] Sea $G = \mathbb{Z}_{2579}^x$ el grupo multiplicativo de unidades módulo $p = 2579$. Como $d = \phi(2579) = 2578 = 2 \cdot 1289$, con 2 y 1289 primos, y $2^2 = 4$ y $2^{1289} = -1$ en $\mathbb{Z}_{2579}^x$, ambos distintos de 1, concluimos que $g = 2 \in G$ es el generador de G. Una descripción de G equivale a unos bits diciendo: "G es de la forma $\mathbb{Z}_p^x$". Entonces, G es identificado como el conjunto $\{0, ..., p-1\}$ y la representación binaria de los enteros proporciona la representación de los elementos de G. En este ejemplo, tomando $n = 12$, ciertas herramientas para aritmética eficiente muestran que las operaciones de grupo pueden realizarse con coste cuadrático, y por tanto, polinómico en n.

Procedemos a describir el intercambio de claves de Diffie-Hellman basándonos en la siguiente representación gráfica.

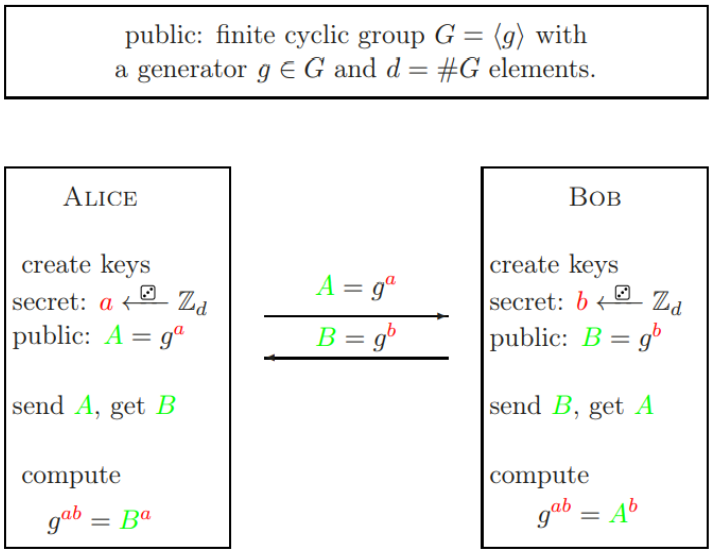

Figura 1.3: Esquema del intercambio de claves Diffie-Hellman [26].

En primer lugar, Alice y Bob deben elegir de forma independiente una clave secreta aleatoria a y $b \in \mathbb{Z}_d$. A continuación, ambos publican sus claves públicas $A = g^a$ y $B = g^b$, respectivamente. Ahora, ambos pueden calcular, tal vez para su uso en un criptosistema de clave simétrica, la clave secreta compartida g^{ab}:

$$g^{ab} = B^a = A^b$$

ya que

$$B^a = (g^b)^a = g^{ba} = g^{ab} = (g^a)^b = A^b$$

El algoritmo es el siguiente:

Algorithm 5 Intercambio de claves Diffie-Hellman

Input: Parámetro de seguridad n.
Output: G, g, d explicadas y clave secreta compartida $k_B = k_A = g^{ab}$.

1: Describir un grupo cíclico finito $G = \langle g \rangle$, siendo g su generador y con $d = \#G$ elementos, donde d es un entero de n bits.
2: Alice elige su clave secreta $sk = a \in \mathbb{Z}_d$, y publica su clave pública $pk = A = g^a \in G$.
3: Bob elige su clave secreta $sk = b \in \mathbb{Z}_d$, y publica su clave pública $pk = B = g^b \in G$.
4: Alice y Bob intercambian sus claves públicas A y B.
5: Alice obtiene su clave secreta compartida $k_A = B^a$ en G.
6: Bob obtiene su clave secreta compartida $k_B = A^b$ en G.
7: Alice y Bob tienen ahora sus claves secretas compartidas $k_B = k_A$.

Ejemplo 1.14. [26] Continuamos con el ejemplo tratado anteriormente. Podemos así seguir con exactitud los pasos vistos en el algoritmo de intercambio de claves:

1: Recordemos que estábamos trabajando con $G = \mathbb{Z}_{2579}^x$ y cuyo generador es $g = 2 \in G$.
2: Alice elige su clave secreta $sk = a = 765 \in \mathbb{Z}_d$, y publica su clave pública $pk = A = g^a = 2^{765} = 949 \in G$.
3: Bob elige su clave secreta $sk = b = 853 \in \mathbb{Z}_d$, y publica su clave pública $pk = B = g^b = 2^{853} = 435 \in G$.
4: Alice y Bob intercambian sus claves públicas $A = 949$ y $B = 435$.
5: Alice obtiene su clave secreta compartida $k_A = B^a = 435^{765} = 2424 \in G$.
6: Bob obtiene su clave secreta compartida $k_B = A^b = 949^{853} = 2424 \in G$.
7: Alice y Bob tienen ahora sus claves secretas compartidas $k_B = k_A = 2424 \in G$.

Observación 1.15. Como ya hemos mencionado, el método es correcto, en cuanto a que tanto Bob como Alice obtienen la misma clave secreta compartida que les permite generar un cifrado simétrico para comunicarse, entre otras cosas. Por otro lado, en cuanto a eficiencia, la operación más costosa es la exponenciación en G, pero justo en el apartado anterior de RSA explicamos una forma de realizar esta operación de manera mucho más eficiente, específicamente en el ejemplo 1.11. Por último, el principal problema que tiene este método

en cuanto a seguridad, es el ataque conocido como *man-in-the-middle* (MITM). En nuestro caso, este ataque consiste en que Eva se hace pasar por Alice cuando habla con Bob y por Bob cuando intercambia datos con Alice. Ambas partes creen que al otro lado tienen a la pareja legítima, pero Eva puede actuar de forma maliciosa generando su propia parte de la clave compartida utilizada. Para solucionar esto, pueden simplemente comparar si las claves secretas compartidas son idénticas. Si no lo son, pueden estar seguros de que alguien ha estado interfiriendo en su comunicación.

Para finalizar, debemos destacar que en la actualidad, el protocolo Diffie-Hellman se emplea en situaciones prácticas ampliamente reconocidas. Un ejemplo de su uso es el protocolo de mensajería instantánea conocido como *Off-the-Record Messaging* (OTR). Además, se utiliza en la red de anonimato denominada *Tor*, donde el protocolo Diffie-Hellman se implementa sobre una conexión TLS previamente establecida en una capa inferior. Esta implementación se utiliza para generar claves de sesión entre el cliente y los nodos de enrutamiento de la red, y estas claves se emplean para cifrar las capas de cebolla de los paquetes que circulan a través de la red.

2 Problema de la mochila

En este capítulo vamos a explicar todos los conceptos necesarios para entender el problema de la mochila, pilar de este trabajo y cuestión que trataremos en capítulos posteriores, ya que será la base del criptosistema en el que nos centraremos. Las principales fuentes bibliográficas han sido [12], [2], [7] y [1].

2.1. Los siete problemas del milenio

Los problemas del milenio son un conjunto de siete enigmas matemáticos que desafían a los científicos desde su formulación en el año 2000 por el Clay Mathematics Institute. Se hizo como un llamado a resolver siete de los problemas matemáticos más complejos de toda la historia. Para motivar más esta búsqueda, se anunció además que la resolución de cada uno de estos problemas sería premiada con una suma de un millón de dólares. Hasta el día de hoy, sólo uno de estos problemas ha sido resuelto, la Conjetura de Poincaré.

A continuación se enumeran los problemas del milenio acompañados de una breve descripción de cada uno [12]:

1. **La conjetura de Hodge**: Es un problema relacionado con la geometría y la topología algebraica. Se centra en entender la estructura de ciertos espacios matemáticos que surgen al estudiar variedades algebraicas y sus propiedades geométricas. Resolver esta conjetura permitiría un mayor entendimiento de la relación entre el álgebra y la geometría.
2. **La conjetura de Poincaré**: Plantea que la esfera cuatridimensional (3-esfera) es la única variedad compacta cuatridimensional en la que todo lazo (1-esfera) se puede deformar en un punto. Este es el único problema que ha sido resuelto hasta el momento.
3. **La hipótesis de Riemann**: Propone que todos los ceros no triviales de la función zeta de Riemann tienen parte real $\frac{1}{2}$. Resolverla implicaría grandes avances en la estimación de errores en matemática aplicada y distribución de primos en el campo del análisis matemático.
4. **Las ecuaciones de Navier-Stokes**: El enunciado del problema es demostrar si a partir de unas condiciones iniciales de fluido laminar, la solución del flujo para todos los instantes de tiempo es también un flujo laminar. Estas ecuaciones en derivadas parciales describen el comportamiento de los fluidos. Su resolución permitiría entender

mejor la mecánica de fluidos necesaria para el diseño de aeronaves y embarcaciones, entre otras cosas.

5. **Existencia de Yang-Mills y del salto de masa**: En teoría cuántica de campos, la teoría de Yang-Mills, que generaliza la teoría de Maxwell del campo electromagnético, ha sido utilizada para describir la cromodinámica cuántica, que explica la estructura de los bariones (como son los protones o neutrones) y el grado de estabilidad del núcleo atómico. Aplicando la teoría clásica de campos, aparecen soluciones que describen partículas sin masa. Sin embargo, el fenómeno denominado *confinamiento de carga de color*, unicamente permite estados formados por partículas muy masivas.

6. **La conjetura de Birch y Swinnerton-Dyer**: Esta conjetura está relacionada con las curvas elípticas. Plantea que existe una forma sencilla de saber al caso si esas ecuaciones tienen un número finito o infinito de soluciones racionales. Resolver esta conjetura tendría implicaciones significativas en la teoría de números y la aritmética algebraica.

El séptimo problema de la lista es el problema **P vs NP**. Sin duda, este es el problema que más nos incumbe y, por lo tanto, le daremos más atención que a los demás. No obstante, antes de enunciarlo, es necesario realizar algunas definiciones importantes para comprenderlo adecuadamente.

2.2. Problemas NP-completos

La *teoría de la complejidad* [7] es un subcampo de la informática teórica cuyo objetivo principal es clasificar y comparar la dificultad práctica de resolver problemas sobre objetos combinatorios finitos. En otras palabras, estudia el crecimiento computacional (principalmente en memoria y tiempo) de resolver un determinado problema, en relación a lo que crece el tamaño de dicho problema.

Imaginemos que queremos ordenar un vector formado por n elementos. Es evidente que a nuestro ordenador le llevará más tiempo ordenar 1000 números que ordenar 100. Pero, ¿cuánto tiempo más necesitará? Esta relación es lo que se estudia en teoría de la complejidad. Un punto importante es que el tiempo necesario para ejecutar un programa, dependerá del algoritmo que utilicemos, por lo que debemos definir algún término que permita medir la eficiencia de dicho algoritmo.

La *complejidad de un algoritmo* es la relación computacional que describe la cantidad de tiempo necesario para ejecutar dicho algoritmo. Diremos que la *complejidad de un algoritmo* es lineal, o que el *algoritmo se resuelve* en tiempo lineal, cuando la función del tiempo en ejecutar el algoritmo respecto al tamaño del problema sea una función lineal. Puede ocurrir que esta función sea logarítmica, polinómica, o incluso exponencial. Aplicando este concepto, definimos la *complejidad de un problema* como la complejidad del mejor algoritmo que lo resuelve.

La teoría de la complejidad intenta hacer distinciones entre problemas proponiendo un criterio formal de lo que significa que un problema matemático sea "NP-completo". Para entender este concepto de manera precisa, es necesario dar un modelo formal de ordenador.

Una *máquina de Turing* [30] es una máquina informática teórica inventada por Alan Turing para servir como un modelo idealizado para el cálculo matemático. Consta de una línea de celdas conocida como *cinta* que se puede mover hacia adelante y hacia atrás, un elemento activo conocido como *cabeza* que posee una propiedad conocida como *estado* y que puede cambiar la propiedad conocida como *color* de la celda activa debajo de ella, y un conjunto de instrucciones sobre cómo la cabeza debe modificar la celda activa y mover la cinta. Tras conocer este nuevo concepto, veamos su definición formal.

Definición 2.1. [1] Una *máquina de Turing* es un modelo computacional que de manera automática realiza operaciones de lectura y escritura sobre una entrada llamada cinta, generando una salida en esta misma. Definimos una maquina de Turing con una sola cinta como la siguiente 7-tupla:

$$M = (Q, \Sigma, s, b, F, \Gamma, \partial)$$

- Q es un conjunto finito de estados.
- Σ es un conjunto finito de símbolos, denominado alfabeto de máquina o de entrada.
- $s \in Q$ es el estado inicial.
- $b \in \Gamma$ es el único símbolo que se puede repetir un número infinito de veces. Se denomina símbolo blanco.
- $F \subseteq Q$ es el conjunto de estados finales de aceptación.
- Γ es un estado finito de símbolos de cinta, denominado alfabeto de cinta ($\Sigma \subseteq \Gamma$).
- $\partial : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ es una función parcial denominada función de transición, donde L es el movimiento a la izquierda y R es el movimiento a la derecha.

Inicialmente esta máquina fue definida por Alan Turing como "máquina automática" en 1936. Fue tras su muerte cuando adopto el término usado a día de hoy. La importancia de las máquinas de Turing a lo largo de la historia es doble. En primer lugar, fue uno de los primeros modelos teóricos de las computadoras, ya que puede ser adaptada para simular la lógica de cualquier algoritmo de computación y es particularmente útil en la explicación de las funciones de una CPU dentro de un ordenador. Por otro lado, ha servido de base para un gran desarrollo teórico en ciencas de la computación y teoría de la complejidad. Una vez explicado todo esto, procedemos a dar la definición de un tipo particular de máquina de Turing.

Definición 2.2. [27] Diremos que una máquina de Turing es *no determinista* si tiene la capacidad de seguir múltiples rutas computacionales de manera simultánea, con la limitación de que estas rutas no pueden comunicarse entre sí.

Tras todas estas definiciones, ya tenemos los recursos necesarios para afrontar el problema P vs NP.

Definición 2.3. [28] Denominamos *NP* al conjunto de problemas que podemos resolver mediante una máquina de Turing no determinista en tiempo polinómico, equivalentemente, son los problemas en los que podemos comprobar en tiempo polinomial si una respuesta es correcta o no.

Definición 2.4. [29] Denominamos *P* al conjunto de problemas de los que se conoce al menos un algoritmo que permite resolver el problema en tiempo polinómico.

Está claro que si de un problema conocemos un algoritmo que permite resolverlo en tiempo polinomial, entonces también podemos verificar si una solución es correcta o no en tiempo polinomial, es decir, se puede resolver mediante una máquina de Turing no determinista. Por tanto, sabemos que P $\subseteq$ NP. Lo que no es tan claro es la inclusión contraria. Esta nos dice que, si podemos comprobar una solución en tiempo polinomial, podemos entonces encontrar una solución en tiempo polinomial. El siguiente ejemplo ilustra cómo, desde una perspectiva intuitiva, parece que P $\neq$ NP. Sin embargo, hasta que alguien resuelva este importante problema, no podremos saber con certeza la respuesta a esta pregunta: ¿Es el conjunto P = NP?

Ejemplo 2.5. Supongamos que nos dan un conjunto de números positivos y negativos $A = \{+1, -3, +7, -4, -2, -7, +9, -3, -5, -1. - 8\}$, y nos preguntan si podemos obtener el conjunto $X \subseteq A$ tal que la suma de sus valores sea cero. Si nos dan la respuesta $X = \{-3, -4, +7\}$, podemos comprobar facilmente si es solución o no, simplemente sumando los valores de X y viendo si su resultado es igual a cero. Así, $-3 - 4 + 7 = 0$, por lo que sí que es solución. No obstante, encontrar ese conjunto de entre todos los posibles no parece una tarea tan sencilla.

Como definición principal de este apartado, explicamos a continuación el concepto de problema "NP-completo", aunque para ello necesitamos una última definición.

Definición 2.6. Llamamos *reducción* a una transformación en tiempo polinomial, de un problema de decisión en otro equivalente. Esto es, sea A el conjunto de instancias del primer problema, y B el conjunto de instancias del segundo, definimos una reducción r como $r : A \to B$ tal que $a \in A$ es sí $\iff$ $r(a) \in B$ es sí, y además se realiza en tiempo polinomial.

Definición 2.7. Diremos que un problema es *NP-completo* si es un problema de decisión perteneciente a NP, que además verifica que existe una reducción de cada problema de NP a él.

Los ejemplos de problemas NP-completos incluyen el "ciclo hamiltoniano", "el problema del viajante de comercio" y "el problema de la mochila".

2.3. Enunciado del problema

El problema de la mochila, también conocido como "Knapsack problem" en inglés, es un problema del campo de la teoría de algoritmos cuyo problema de decisión asociado es NP-completo. Se trata de un desafío donde se busca la mejor manera de almacenar objetos en una mochila, maximizando el valor total de los elementos llevados, sin exceder la capacidad de carga máxima de la mochila.

Imaginemos que tenemos una mochila con una capacidad máxima para cargar un cierto peso determinado. Además, disponemos de un conjunto de elementos, cada uno con un peso y un valor asociado. El objetivo es seleccionar qué elementos incluir en la mochila de manera que la suma de los valores sea lo más alta posible, pero sin sobrepasar la capacidad máxima de carga.

Definición 2.8. [18, 2] Sean $a = \{a_1, ..., a_n\} \subseteq \mathbb{N}^*$ un *conjunto de pesos* y $S \in \mathbb{N}^*$ la *capacidad total*, el *problema de la mochila* busca encontrar el *vector de soluciones* $x = \{x_1, ..., x_n\} \subseteq \mathbb{N}^*$ que maximice el valor de la mochila y verifique:

$$\sum_{i=1}^{n} a_i \cdot x_i \leq S$$

Observación 2.9. En realidad, nosotros nos centraremos calcular la solución del caso específico del problema de la mochila en el que se cumple la condición de igualdad:

$$\sum_{i=1}^{n} a_i \cdot x_i = S$$

Veamos un ejemplo que aclare las ideas.

Ejemplo 2.10. Sea $a = (6, 4, 9, 5)$ y la capacidad máxima de la mochila $S = 17$, entonces existe una solución al problema de la mochila dada por $x = (2, 0, 0, 1)$, ya que verifica:

$$6 \cdot 2 + 4 \cdot 0 + 9 \cdot 0 + 5 \cdot 1 = 17$$

Pero además, esta solución no es única, porque podemos encontrar otra solución de la forma $x = (0, 2, 1, 0)$, pues:

$$6 \cdot 0 + 4 \cdot 2 + 9 \cdot 1 + 5 \cdot 0 = 17$$

Dado que se cree que P $\neq$ NP, podemos asumir que es un problema computacionalmente difícil de resolver en general, a menos que se tenga acceso a la "información trampilla" utilizada en su diseño. Dado que sólo el diseñador puede resolver fácilmente este problema, la idea es que otros usuarios puedan enviarle información oculta en la solución sin temor a que un atacante sea capaz de extraerla.

Existen diferentes variantes del problema de la mochila, tales como el ejemplo de la mochila de capacidad fraccionaria (donde se pueden tomar porciones de elementos) o la mochila de $0/1$ (donde se pueden o no tomar los elementos completos). Será este último caso, también llamado *problema de la suma de subconjuntos*, el que trataremos más adelante gracias al hecho de que su vector solución está formado por valores binarios. Es decir, el problema de la mochila de $0/1$ es igual al definido previamente, salvo que su solución $x = \{x_1, ..., x_n\} \in \{0, 1\}$. Veamos un ejemplo de esta variante:

Ejemplo 2.11. [23] Sea $a = (4, 3, 9, 1, 12, 17, 19, 23)$ y la capacidad máxima de la mochila $S = 35$, entonces existe una solución al problema de la mochila y está dada por $x = (0, 1, 0, 1, 1, 0, 1, 0)$, pues se verifica que:

$$4 \cdot 0 + 3 \cdot 1 + 9 \cdot 0 + 1 \cdot 1 + 12 \cdot 1 + 17 \cdot 0 + 19 \cdot 1 + 23 \cdot 0 = 35$$

Por el contrario, si la capacidad máxima fuese $S = 6$, el problema no tendría solución.

A lo largo de los años, se han desarrollado varios algoritmos y enfoques para abordar el problema de la mochila, cada uno con sus ventajas y desventajas. Algunos de estos algoritmos proporcionan soluciones exactas, mientras que otros se enfocan en obtener soluciones aproximadas que puedan ser computadas de manera más eficiente.

Este problema tiene diversas aplicaciones en la vida real, como la optimización de recursos en logística, el manejo de recursos limitados en la planificación de proyectos o la asignación de tareas con restricciones de recursos, entre otros.

3 Criptosistema de Merkle-Hellman

El propósito de este capítulo es abordar de manera clara las diversas variantes del criptosistema de Merkle-Hellman, desde sus fundamentos teóricos hasta sus aplicaciones prácticas. Para lograrlo, haremos uso de los conceptos presentados en los capítulos previos, además de consultar las siguientes fuentes de referencia: [18], [19] y [24].

3.1. Introducción

El criptosistema de Merkle-Hellman es uno de los primeros criptosistemas de clave pública y fue desarrollado en la década de 1970 por Ralph C. Merkle y Martin E. Hellman. Este sistema de clave pública basado en el problema de la mochila, ha cautivado el interés de investigadores y entusiastas de la seguridad durante décadas debido a su simplicidad y elegancia. En este capítulo, describiremos el criptosistema de Merkle-Hellman, desvelando sus principios fundamentales y descubriendo cómo su ingenioso diseño permite el cifrado y descifrado de mensajes.

Nuestra explicación comienza con una clara y concisa descripción de los conceptos clave que sustentan el criptosistema. Posteriormente, explicaremos distintos algoritmos de construcción de claves, donde veremos cómo se forja el candado y la llave que garantizará la confidencialidad de nuestros mensajes. Además, durante el proceso de cifrado, apreciaremos cómo una secuencia aparentemente aleatoria de números, transforma nuestros mensajes en datos ilegibles ante los ojos no autorizados.

A medida que desentrañemos los misterios del criptosistema de Merkle-Hellman, su relevancia en el panorama actual de la ciberseguridad se volverá evidente. Aunque más adelante, destacaremos las debilidades que tiene e introduciremos los ataques más relevantes que ha recibido, tales como el ataque de Adi Shamir, entre otros.

3.2. Preludio al criptosistema

El criptosistema de Merkle-Hellman es uno de los primeros criptosistemas de clave pública que se crearon y el primero que se publicó basado en el problema de la mochila. Sus autores, Ralph C. Merkle y Martin E. Hellman, como ya hemos comentado, lo desarrollaron en la década de los 1970, publicando su artículo concretamente en el año 1978.

En este artículo nos explican que dada una supuesta solución al problema x se puede comprobar fácilmente si es solución o no, en un máximo de n iteraciones, pero se sospecha que encontrar una solución requiere un número de operaciones que crece exponencialmente en n. Esto nos estaría confirmando que se trata de un problema NP-completo y por tanto uno de los problemas computacionales más difíciles de naturaleza criptográfica. Es más, aplicando la fuerza bruta o la búsqueda exhaustiva por ensayo y error de todas las 2^n posibles soluciones x, es computacionalmente inviable su cálculo cuando n es mayor a 100 ó 200.

Los autores explican en su artículo, que en 1978 ya se conocía un método para resolver problemas de mochilas que requería una complejidad de $2^{n/2}$, tanto en tiempo de computación como en memoria de almacenamiento. Asimismo, también explican que R. Schroeppel ideó un algoritmo que requiere una complejidad de $2^{n/2}$ en tiempo y $2^{n/4}$ de espacio. Sin embargo, esto depende en gran medida de la elección de los valores iniciales de a. Procedemos así a realizar la primera definición del capítulo.

Definición 3.1. [19] Diremos que una sucesión $\{a_i\}_{i=1}^n$ es *supercreciente* si verifica que:

$$a_i > \sum_{j=1}^{i-1} a_j \text{ , para } i = 2, ..., n$$

Ejemplo 3.2. La sucesión $\{a_n\} = 1, 3, 8, 16, 30, 64, 143, 270, ...$ forma una sucesión supercreciente puesto que:

$$\begin{aligned} a_2 &= 3 > a_1 = 1 \\ a_3 &= 8 > \sum_{i=1}^{2} a_i = 3 + 1 = 4 \\ a_4 &= 16 > \sum_{i=1}^{3} a_i = 4 + 8 = 12 \\ &\vdots \end{aligned}$$

Diremos que una *mochila con trampilla* es un problema de la mochila en la que una cuidadosa elección de los valores a, permite al diseñador del problema resolver fácilmente cualquier solución x, pero impide que cualquier otro usuario la encuentre.

Distinguimos entonces varios casos según la elección que hagamos de a:

1. Si $a = (1, 2, 4, ..., 2^{n-1})$, entonces resolver x equivale a buscar la representación binaria de S.

2. Si a es un sucesión supercreciente con n elementos, también es fácil hallar x ya que cumple el siguiente algoritmo:

$$x_n = 1 \iff S \geq a_n$$
$$x_i = 1 \iff S \geq a_i + \sum_{j=i+1}^{n} a_j x_j \text{ , para } i = n-1, ..., 1$$

3. Si a es una sucesión de n elementos formada eventualmente por números primos, también es fácil hallar x, ya que los valores de la sucesión a puede que dividan a la capacidad total de la mochila S.

A continuación, vamos a proceder a explicar varios métodos para construir mochilas con trampilla y explicaremos como se utilizan para el envío de información.

Por un lado, el primero de ellos será la búsqueda de *mochilas con trampillas aditivas*. En este primer método, la capacidad total de la mochila S debe expresarse como suma de los valores de la clave pública a. La idea de este tipo de mochilas con trampilla es usar la propiedad que tienen las sucesiones supercrecientes para obtener el mensaje x.

Por otro lado, describiremos también el método de construcción de *mochilas con trampillas multiplicativas*. En este caso, la capacidad total de la mochila S se expresa como producto de valores del vector a', que está formado eventualmente por números primos. Un punto importante de este apartado, es que una mochila multiplicativa se transforma en una mochila aditiva tomando logaritmos. Para que ambos vectores tengan valores razonables, tal y como se explica en [20], los logaritmos se toman sobre $GF(m)$, donde m es el número primo tomado como clave privada. La idea es obtener S' y utilizar los valores primos del vector a' para obtener el mensaje x.

Finalmente, describiremos el *método iterativo* que consiste en una combinación de los métodos vistos. En este caso, la idea es aplicar reiteradas veces uno o ambos de los métodos explicados, conseguiendo de esta forma aumentar la seguridad del mensaje x.

3.3. Descripción del criptosistema

En primer lugar dejemos claro nuestro objetivo: buscamos un método para establecer una comunicación segura y confidencial entre dos partes, en la que se garantice que los mensajes transmitidos no puedan ser leídos ni interpretados por terceros no autorizados.

Con esto en la cabeza, comencemos la explicación de [18]. La idea es que un usuario I genere un vector mochila con trampilla $a(I)$. Esto no es más que una clave que él puede utilizar para encontrar fácilmente la solución, que usará como clave pública. Cuando

otro usuario J quiera mandarle un mensaje x al usuario I, deberá enviarle el mensaje S encriptado, usando la clave pública de I. Por tanto, J debe enviar el mensaje cifrado $S = x \cdot a(I)$. De esta forma, el destinatario I puede recuperar x a partir de S, gracias a $a(I)$, pero nadie más podrá. Recordemos que habíamos escogido un vector mochila con trampilla $a(I)$ justamente para que el usuario I sí que pueda encontrar la solución x a partir de S. Procedemos por tanto, a explicar el método para construir mochilas con trampilla aditivas, multiplicativas y mediante iteración.

3.3.1. Método de construcción de mochilas con trampilla aditivas

En primer lugar, el diseñador debe elegir dos números, m y w tal que w sea invertible módulo m, o lo que es lo mismo, que $gcd(w, m) = 1$; o que m y w sean coprimos. Es crucial que estos dos números sean secretos y sólo los conozca el propio diseñador.

Los datos iniciales estan formados por los dos valores previos m y w, y una sucesión supercreciente a' generada por el diseñador, también secreta y conocida únicamente por él. El propósito a continuación, es descifrar el mensaje x a partir de la información recibida $S = a \cdot x$. Sin embargo, la idea es aplicar una transformación al mensaje recibido para obtener $S' = a' \cdot x'$. De esta forma, una vez conozcamos S', podremos hayar el mensaje x. La ventaja de esta transformación es que al ser a' una sucesión supercreciente, permitirá al diseñador resolver el problema de manera más sencilla y eficiente.

Mostramos este pequeño esquema a modo de resumen:

> Datos generados: m, w (valores coprimos) y a' (sucesión supercreciente)
>
> Datos recibidos: S (mensaje cifrado)
>
> Datos objetivo: a (clave pública) y S' (mensaje transformado)

Para ello, el diseñador usará los valores m, w y a' para obtener la clave pública a, que será revelada para que cualquier persona que desee enviarnos mensajes pueda utilizarla para cifrarlos; y usará m, w y S para obtener S', que representa la alteración del mensaje recibido.

Vamos a explicar como obtener la clave pública a. Para ello, el diseñador debe elegir un vector a', de tamaño n, que sea una sucesión supercreciente. Ahora, debe transformar el vector a' en un vector mochila con trampilla a de la siguiente forma:

$$a_i \equiv w \cdot a'_i \bmod m \text{ , con } i = 1, ..., n$$

Como los a_i están distribuidos pseudoaleatoriamente, cualquiera que conozca a pero no conozca ni w ni m, tendrá grandes dificultades para resolver un problema de mochila que incluya a. En cambio, el diseñador sí que podrá calcular S' de manera sencilla:

$$\begin{aligned} S' &= w^{-1} \cdot S \bmod m \\ &= w^{-1} \sum_{i=1}^{n} x_i \cdot a_i \bmod m \\ &= w^{-1} \sum_{i=1}^{n} x_i \cdot w \cdot a'_i \bmod m \\ &= \sum_{i=1}^{n} x_i \cdot a'_i \bmod m \end{aligned}$$

Así, si m verifica que $m > \sum_{i=1}^{n} a'_i$, entonces se verifica que $S' = \sum_{i=1}^{n} x_i \cdot a'_i$.

Finalmente, hemos logrado la transformación del problema en la forma $S' = a' \cdot x$, lo que nos facilita enormemente la resolución de x debido a que a' es una sucesión supercreciente y podemos aplicar sus propiedades vistas en el apartado 3.2. Esto es exactamente lo que estábamos buscando desde el principio y, por supuesto, coincide con el mensaje original.

Mostramos ahora un ejemplo con un tamaño muy pequeño con el objetivo de aclarar las ideas aquí explicadas.

Ejemplo 3.3. Sea $n = 5$. Tomemos $m = 2113$, $w = 988$ y $a' = (3, 42, 105, 249, 495)$. Entonces, podemos calcular $w^{-1} = 802$, con la operación inverso modular y gracias a la operación vista antes, obtenemos $a = (851, 1349, 203, 904, 957)$.

Ahora, supongamos que un usuario quisiera mandarnos el mensaje $x = (0, 0, 0, 1, 1)$. Para ello, ese usuario emisor debe realizar el producto escalar de x, con nuestra clave pública a, obteniendo $S = 1861$. Así, una vez que el emisor nos envíe el valor de S, el diseñador ya podrá calcular S' de la siguiente forma:

$$\begin{aligned} S' &= w^{-1} \cdot S \bmod m \\ &= 802 \cdot 1861 \bmod 2113 \\ &= 744 \end{aligned}$$

Por tanto, como $S' = a' \cdot x$ y a' es una sucesión supercreciente, aplicamos lo visto a este tipo de suceciones:

$$\begin{aligned} &S' = 744 \geq a'_5 = 495 \Rightarrow x_5 = 1 \\ &S' = 744 \geq a'_5 + a'_4 = 744 \Rightarrow x_4 = 1 \\ &S' = 744 \ngeq a'_5 + a'_4 + a'_3 = 849 \Rightarrow x_3 = 0 \\ &\qquad \vdots \end{aligned}$$

Tras realizar esto sucesivamente, obtenemos como solución $x = (0, 0, 0, 1, 1)$, que era justamente el mensaje original. Podemos comprobarlo viendo que se verifica:

$$S = a \cdot x = 851 \cdot 0 + 1349 \cdot 0 + 203 \cdot 0 + 904 \cdot 1 + 957 \cdot 1 = 1861$$

En este ejemplo se puede ver como cualquiera que no conozca los valores de m, w y a' tiene grandes problemas para obtener el mensaje x, aunque el vector a sea público para todo el mundo.

Observación 3.4. Merkle y Hellman indican en su artículo [18] los valores que ellos recomiendan y consideran seguros para este criptosistema. Son los siguientes:

- $n = 100$
- $m \in [2^{201} + 1, 2^{202} - 1]$
- $a'_i \in [(2^{i-1} - 1) \cdot 2^{100} + 1, 2^{i-1} \cdot 2^{100}]$, con $i = 1, ..., 100$
- $w' \in [2, m - 2]$.

Luego w' debe dividirse por el máximo común divisor de w' y m para para obtener w.

Además, explican en su publicación que estas elecciones garantizan que se cumpla la siguiente condición: $m > \sum_{i=1}^{n} a'_i$, y que en caso de que alguien intente descifrarlo, tenga al menos 2^{100} posibilidades por cada parámetro, lo que hace que sea muy difícil descrifrarlo o incluso intentar encontrar uno solo de estos parámetros.

3.3.2. Método de construcción de mochilas con trampilla multiplicativas

Al igual que en el método anterior de las mochilas con trampilla aditiva, el diseñador en primer lugar debe elegir dos números coprimos m y b. Vuelve a ser crucial que estos dos números sean secretos y sólo los conozca el propio diseñador.

De manera muy similar al método anterior, el propósito es descubrir el mensaje x. Por tanto, se debe transformar el sistema $S = a \cdot x$ en otro donde a' actúe como una sucesión con valores eventualmente primos de la siguiente forma: $S' = a' \cdot x'$. Esta transformación permitirá resolver el problema de manera más sencilla y eficiente.

Mostramos este pequeño esquema a modo de resumen:

> Datos generados: m, b (valores coprimos) y a' (sucesión de primos eventuales)
> Datos recibidos: S (mensaje cifrado)
> Datos objetivo: a (clave pública) y S' (mensaje transformado)

Para ello, el diseñador usará los valores m, b y a' para obtener la clave pública a, y podrá además, obtener S' a partir de m, b y S. Expliquemos primero cómo obtener la clave pública. Para ello, se debe elegir un vector a', de tamaño n, que esté formado eventualmente por

números primos. Posteriormente, se debe transformar este vector a' en un vector mochila con trampilla a, tal que cada a_i verifique lo siguiente:

$$\begin{gathered} b^{a_i} \bmod m = a'_i \\ \Updownarrow \\ b^{a_i} = y \cdot m + a'_i \\ \Updownarrow \\ a_i \cdot ln(b) = ln(y \cdot m + a'_i) \\ \Updownarrow \\ a_i = \frac{ln(y \cdot m + a'_i)}{ln(b)} \end{gathered}$$

con $y \in \mathbb{N}$, $i = 1, ..., n$ y $ln(b) \neq 0$, es decir, $b \neq 1$.

Encontrar logaritmos sobre $GF(m)$ es relativamente fácil si $m - 1$ sólo tiene factores primos pequeños. A continuación, para calcular S', debemos aplicar:

$$\begin{aligned} S' &= b^S \bmod m \\ &= b^{\sum a_i \cdot x_i} \bmod m \\ &= \prod_{i=1}^{n} b^{a_i \cdot x_i} \bmod m \\ &= \prod_{i=1}^{n} a_i^{'x_i} \bmod m \end{aligned}$$

Así, es necesario que se verifique $m > \prod a'_i$, para asegurar que $\prod a'_i \cdot x_i \bmod m = \prod a'_i \cdot x_i$.

Finalmente, hemos logrado la transformación del problema en la forma $S' = a' \cdot x$, lo que nos facilita enormemente la resolución de x debido a que a' es una sucesión con valores eventualmente primos y podemos descomponer S' en los valores de esta sucesión. Esto es exactamente lo que estábamos buscando desde el principio y de nuevo, coincide con el mensaje original.

Mostramos a continuación un pequeño ejemplo que nos ayude a aclarar las ideas explicadas.

Ejemplo 3.5. Sea $n = 5$. Tomamos $m = 257$, $b = 131$ y $a' = (2, 3, 5, 7, 11)$. Entonces, calculamos la clave pública a, de la forma ya explicada:

$$\begin{aligned}
&131^{80} \bmod 257 = 2 \Rightarrow a_1 = 80\\
&131^{183} \bmod 257 = 3 \Rightarrow a_2 = 183\\
&131^{81} \bmod 257 = 5 \Rightarrow a_3 = 81\\
&131^{195} \bmod 257 = 7 \Rightarrow a_4 = 195\\
&131^{28} \bmod 257 = 11 \Rightarrow a_5 = 28
\end{aligned}$$

Obtenemos entonces $a = (80, 183, 81, 195, 28)$. Ahora, supongamos que un usuario quisiera mandarnos el mensaje $x = (0, 1, 1, 0, 1)$. Para ello, ese usuario emisor debe realizar el producto escalar de x, con nuestra clave pública a, obteniendo $S = 292$. De esta forma, una vez que el emisor nos envíe el valor de S, el diseñador ya podrá calcular S' gracias a la información de trampilla m y b de la siguiente forma:

$$\begin{aligned}
S' &= b^S \bmod m\\
&= 131^{292} \bmod 257\\
&= 165\\
&= 2^0 \cdot 3^1 \cdot 5^1 \cdot 7^0 \cdot 11^1
\end{aligned}$$

Esto implica que $x = (0, 1, 1, 0, 1)$, que era justamente el mensaje original. Podemos comprobarlo viendo que se verifica:

$$S = a \cdot x = 80 \cdot 0 + 183 \cdot 1 + 81 \cdot 1 + 195 \cdot 0 + 28 \cdot 1 = 292$$

En este ejemplo se puede ver como cualquiera que no conozca los valores de m, b y a' tiene grandes problemas para obtener el mensaje x, aunque el vector a sea público para todo el mundo.

Observación 3.6. Merkle y Hellman indican en su artículo [18] los valores que ellos recomiendan y consideran seguros para este criptosistema. Tomando $n = 100$, cada a'_i es un número primo de 100 bits de tamaño, por lo que m debe tener tamaño de 10000 bits aproximados para que se verifique $m > \prod a'_i$. En realidad, se puede verificar la condición incluso si m tiene un tamaño de 730 bits, pero recomiendan el valor de 10000 bits.

3.3.3. Método iterativo

Recordemos que en el primer método de mochilas con trampilla explicado, transformábamos un problema de la mochila aparentemente sencillo a' en uno más complicado a. En este caso, para generar un nuevo vector, usaremos la siguiente expresión:

$$a'_i \equiv w^{-1} \cdot a_i \bmod m$$

La idea, como explicábamos en el apartado 3.3.1, es que podemos resolver el problema que envuelve a a porque podemos resolver el problema que envuelve a a'. Sin embargo, en vez de exigir que a' sea una sucesión supercreciente, como imponíamos en el primer apartado, basta con exigir que a' sea transformable en una sucesión supercreciente a'' según la transformación:

$$a''_i \equiv w'^{-1} \cdot a'_i \bmod m'$$

donde el nuevo problema de la mochila obtenido a'', es una sucesión supercreciente igualmente fácil de resolver. Habiendo aplicado esta transformación dos veces, no hay problema en aplicarla una tercera, una cuarta o una quinta vez. Así, es claro que podemos repetir este proceso tanto como queramos. A cada transformación que aplicamos, la estructura del vector a conocido públicamente se vuelve más "oscura".

En esencia, estamos encriptando el vector simple de la mochila mediante la aplicación repetida de una transformación que preserva la estructura básica del problema. Finalmente, el último resultado obtenido es aparentemente una colección de números aleatorios que enmascara totalmente el hecho de que el problema puede resolverse de manera sencilla. Veamos un ejemplo de este método para entenderlo.

Ejemplo 3.7. Para que resulte más sencillo, escogeremos los mismos valores que en el ejemplo 3.3, solo que aplicaremos dos iteraciones. Por tanto, tomamos $n = 5$, $m'' = 2113$, $w'' = 988$ y $a''' = (3, 42, 105, 249, 495)$. Por organización, mostramos la clave privada a cada iteración. En este caso, está formada por:

$$\begin{aligned} m'' &= 2113 \\ w'' &= 988 \\ a''' &= (3, 42, 105, 249, 495) \end{aligned}$$

A continuación, obtenemos el inverso multiplicativo $w''^{-1} = u'' = 802$. Este valor es necesario para obtener el siguiente vector a'', que se calcula según la fórmula:

$$a''_i \equiv u'' \cdot a'''_i \bmod m'' \tag{3.1}$$

Aplicandola, obtenemos $a'' = (293, 1989, 1803, 1076, 1859)$. Con esto, hemos concluido la primera iteración. Generamos a continuación los nuevos valores m y w, que son: $m' = 9889$, $w' = 9662$. La clave privada de esta iteración es por tanto:

$$\begin{aligned} m' &= 9889 \\ w' &= 9662 \\ a'' &= (293, 1989, 1803, 1076, 1859) \end{aligned}$$

Tras calcular el inverso modular $u' = 4095$, volvemos a aplicar la expresión (3.1), obteniendo $a' = (3266, 6308, 6091, 5615, 7964)$. Finalmente debemos obtener la clave pública, por lo que

debemos generar nuevos m y w. Así, $m = 33418$ y $w = 1025$. La nueva clave privada es:

$$\begin{aligned} m &= 33418 \\ w &= 1025 \\ a' &= (3266, 6308, 6091, 5615, 7964) \end{aligned}$$

Por último, tras calcular $u = 2217$, obtenemos que la clave pública es:

$$a = (22434, 16112, 2875, 16959, 11484)$$

Tras este proceso de generación de claves, supongamos que nos quieren enviar el mensaje $(0, 0, 0, 1, 1)$. Como ya sabemos, el emisor debe cifrarlo con la clave pública a, por lo que obtendrá $S = 28443$. Para descifrarlo, debemos aplicar las claves privadas de manera inversa hasta obtener el mensaje original. Así, comenzamos aplicando la última clave privada al valor S recibido:

$$\begin{aligned} S' &= S \cdot w \bmod m \\ &= 28443 \cdot 1025 \bmod 33418 = 13579 \end{aligned}$$

Reiterando este proceso por cada clave privada, obtenemos:

$$\begin{aligned} S'' &= S' \cdot w' \bmod m' \\ &= 13579 \cdot 9662 \bmod 9889 = 2935 \\ S''' &= S'' \cdot w'' \bmod m'' \\ &= 2935 \cdot 988 \bmod 2113 = 744 \end{aligned}$$

Por último, debemos aplicar el proceso de desencriptado explicado en el ejemplo de mochilas aditivas al valor S''' obtenido. El resultado final es $(0, 0, 0, 1, 1)$, que coincide con el mensaje original enviado.

Tras este ejemplo, llegamos a una importante definición, con la que debemos familiarizarnos ya que trataremos con ella durante todo el resto del trabajo.

Definición 3.8. [15] Sea $a = (a_1, ..., a_n)$ un vector de pesos, definimos la *densidad del vector* a como:

$$d(a) = \frac{n}{log_2(max_i a_i)} \text{ , con } i = 1, ..., n$$

En términos de criptosistemas de mochila con clave pública, este valor es una medida aproximada de la tasa de información a la que se transmiten los bits, es decir:

$$d(a) \cong \frac{\text{Número de bits en el mensaje original}}{\text{Media de número de bits en el mensaje cifrado}}$$

Así, definimos la *densidad de un problema* como la densidad de su vector solución.

Más adelante (en el capítulo 5), veremos los ataques por baja densidad basados en el algoritmo L^3. Por ahora, debemos centrarnos en que por cada iteración que se realiza con este método, la densidad del problema de la mochila disminuye. Esto permite realizar ataques por baja densidad como el de Lagarías o Coster. En este punto, es interesante realizar la siguiente definición para entender el concepto de composición de cifrados por sustitución simple y sus diferencias con respecto al método iterativo.

Definición 3.9. [11] Un *cifrado por sustitución simple* es un método de cifrado en el que un simple caracter de un texto es sustituido por otro caracter determinado del alfabeto de sustitución. Esto es, se establecen parejas en la que el segundo elemento de la pareja es el caracter que sustituye al primero. Los alfabetos usados para el texto original y el texto cifrado pueden ser los mismos, aunque no tiene por qué.

El efecto de repetir el proceso iterativo varias veces es muy distinto al obtenido por métodos como el cifrado por sustitución simple. La diferencia es que la repetición de este tipo de cifrados no oculta ni enmascara el mensaje, ya que la composición de cifrados por sustitución es otro cifrado por sustitución. En cambio, la transformación (w, m) no tiene esta propiedad de cierre. Veamos a continuación un contraejemplo que nos mostrará que la repetición de dos transformaciones (w, m) y (w', m'), no es necesariamente equivalente a otra transformación $(\widetilde{w}, \widetilde{m})$.

Demostración. Sea $n = 3$, $m' = 47$, $w' = 17$, $m = 89$, $w = 3$ y $a'' = (1, 5, 10)$, entonces, utilizando el método visto en el apartado 3.3.1 con el vector a'' y los valores w' y m', obtenemos $a' = (17, 38, 29)$. Aplicando de nuevo el mismo método pero con el vector a' obtenido, y los valores w y m, terminamos obteniendo $a = (51, 25, 87)$.

Suponemos ahora que existen valores $\widetilde{w}$ y $\widetilde{m}$ que verifican:

$$a_i = \widetilde{w} \cdot a_i'' \bmod \widetilde{m}$$

Tomando los primeros valores $a_1 = 51$ y $a_1'' = 1$, sustituimos obteniendo:

$$\begin{aligned} 51 &= \widetilde{w} \bmod \widetilde{m} \\ 51 \cdot 5 &= \widetilde{w} \cdot 5 \bmod \widetilde{m} \\ 255 &= \widetilde{w} \cdot 5 \bmod \widetilde{m} \end{aligned} \tag{3.2}$$

Pero ahora, teniendo en cuenta la relación entre los segundos valores $a_2 = 25$ y $a_2'' = 5$,

$$25 = \widetilde{w} \cdot 5 \bmod \widetilde{m}$$

Por tanto, $255 = 25 \bmod \widetilde{m} \Rightarrow 230 = 0 \bmod \widetilde{m} \Rightarrow \widetilde{m} = 230$. Sustituimos este nuevo valor en la ecuación (3.2) anterior:

$$51 = \widetilde{w} \bmod 230$$

Por lo que obtenemos que $\widetilde{w} = 51$. Aunque, si $\widetilde{m} = 230$ y $\widetilde{w} = 51$, entonces usando la relación entre los últimos valores, $a_3 = 87$ y $a_3'' = 10$, llegamos a que:

$$\begin{aligned} 87 &= 51 \cdot 10 \bmod 230 \\ &= 510 \bmod 230 \\ &= 50 \end{aligned}$$

lo cual es una contradicción. Finalmente concluimos que tales valores $\widetilde{w}$ y $\widetilde{m}$ no pueden existir. □

Podrían utilizarse otro tipo de vectores mochilas con trampilla fáciles de resolver, como la mochila con trampilla multiplicativa que hemos visto anteriormente. De esta forma, se pueden combinar ambos métodos en uno solo, presumiblemente más difícil de romper.

3.4. Fichero Público

Como hemos descrito anteriormente, todo este proceso explicado tiene el objetivo de permitir la comunicación segura entre dos usuarios. Así, un usuario I debe situar su clave pública $a(I)$ en una carpeta pública o algún sitio donde el resto de usuarios que quieran enviarle un mensaje tengan acceso. De esta forma, el usuario J puede encontrar $a(I)$ y mandar su mensaje x escondido como $S = a(I) \cdot x$.

Para eliminar el problema de almacenamiento de la clave $a(I)$, J podría preguntarle al usuario I por su clave pública. El problema es que a no ser que J tenga un método para comprobar $a(I)$, otro usuario K puede engañar a J enviandole $a(K)$ y diciendo que ese valor es $a(I)$, por lo que K obtendría el mensaje x. En consecuencia, se requiere un procedimiento que le asegure a J que ha recibido el verdadero valor de $a(I)$.

Dado que se trata de un archivo de acceso público, la solución sería permitir que cada usuario pueda incorporar su propio vector personal al archivo. Luego, una vez autenticados en el sistema, deberían distinguirse de otros usuarios demostrando su capacidad para descifrar mensajes ocultos utilizando ese vector personal. Para finalizar, en cuanto a seguridad del archivo, es evidente que el archivo en sí debe contar con protección contra escritura. Veamos a continuación unos conceptos clave relacionados con este tema.

Definición 3.10. [24, 14] Una *función hash* es una función computable mediante un algoritmo que tiene la siguiente forma:

$$\begin{aligned} H : U &\to M \\ x &\to h(x) \end{aligned}$$

donde la entrada x suele ser una cadena de texto, mientras que la salida $h(x)$ suele ser una cadena de longitud finita, denominada *valor hash*. La característica principal de estas

funciones es que el *valor hash* tiene siempre un tamaño fijo independientemenete de la entrada recibida.

Las propiedades principales de la función hash dependen de la aplicación en particular, pero las más importantes en la práctica son la unidireccionalidad y la resistencia a las colisiones. La primera propiedad indica que es computacionalmente inviable encontrar cualquier entrada tal que al aplicarle una función hash, dé como resultado una salida predefinida (resistencia a la preimagen). Por otro lado, la segunda propiedad nos dice que es computacionalmente inviable encontrar dos valores distintos que produzcan la misma salida (resistencia a la colisión de segunda preimagen), es decir, que no puede haber dos cadenas que tengan el mismo valor hash. Además, una pequeña modificación en la entrada genera un valor hash completamente distinto. Esta definición y sus propiedades mencionadas son necesarias para entender la idea que nos proponen lo autores.

Con el objetivo de preservar la autenticación disminuyendo el tamaño de almacenamiento (aproximadamente se reduce unos 20KB por usuario), Merkle y Hellman recomiendan el uso en la carpeta pública de un hash unidireccional de 100 bits $h[a(I)]$, en lugar de la clave $a(I)$ completa. De esta nueva forma, cuando J recibe $a(I)$ del usuario I, calcula $h[a(I)]$ y comprueba este valor con el almacenado en la carpeta pública. Esta nueva función hash sabemos que debe ser unidireccional y resistente a colisiones, para que el usuario K no pueda generar un vector $a(K)$ que verifique $h[a(K)] = h[a(I)]$.

Al permitir 100 bits para almacenar el nombre del usuario y la dirección, el archivo público contiene ahora 200 bits, en lugar de más de 20 Kbit/usuario. Un sistema con un millón de usuarios requiere un archivo público de 200 millones de bits, en lugar de 20.000 millones de bits que necesitaría de la otra forma. Además un número de 100 bits puede codificarse con 20 caracteres alfanuméricos, lo cual lo hace lo suficientemente pequeño como para caber en una guía telefónica.

Una entrada de la carpeta pública tendría la siguiente forma:

Joe Smith.....497-1573
KSDJR E6K65 3GFVM OMK4K

La segunda linea de la carpeta pública es el hash $h[a(Smith)]$, del vector mochila con trampilla de Smith $a(Smith)$.

4 Ataque de Shamir al criptosistema de Merkle-Hellman

En este capítulo vamos a estudiar el método explicado por Shamir para romper el criptosistema recién explicado de Merkle-Hellman. En realidad, este ataque solo funciona para la ruptura del método básico, pero aun así, supuso el fin para este criptosistema. Nos adentraremos en la idea intuitiva para luego explicar el proceso de manera más formal. La fuente bibliográfica más importante es [22].

4.1. Introducción

El criptosistema de Merkle-Hellman era muy prometedor cuando apareció debido a su bajo coste computacional y su resistencia a la computación cuántica, por lo que era incluso preferido ante su principal competidor, RSA. Así fue hasta que Adi Shamir, en el año 1983, rompió el criptosistema de Merkle-Hellman, dando fin a esta disyuntiva.

Shamir comienza su artículo [22] de la siguiente forma: "El criptosistema Merkle-Hellman es uno de los dos principales criptosistemas de clave pública propuestos hasta ahora. Se demuestra que la variante básica de este criptosistema, en la que los elementos de la clave pública son múltiplos modulares de una secuencia supercreciente, se puede romper en tiempo polinómico". A continuación, procedemos a explicar todo lo necesario para entender la idea que Shamir desarrolló en este documento.

En primer lugar, debemos tener claro que la criptografía es una lucha interminable entre los creadores y los descifradores de código, y este trabajo no pretende dar ninguna respuesta definitiva en este sentido. Es más, el ataque criptoanalítico de Shamir sólo es efectivo para el criptosistema de una iteración propuesto por Merkle y Hellman, que es presumiblemente el menos seguro. Es decir, que este ataque no es aplicable a otros criptosistemas de clave pública propuestos en su artículo, como es el caso del método iterativo.

Destacamos también alguna característica del algoritmo de Shamir, como es su eficiencia incluso en microcomputadores. No obstante, el propio Shamir explica que este algoritmo puede fallar en una clave concreta, aunque las heurísticas indican que estos fallos son extremadamente raros y que están respaldados por cientos de pruebas realizadas con claves de tamaño completo, en las que no se ha cometido ni un solo fallo.

Recordemos primeramente que en el algoritmo básico de una iteración de Merkle-Hellman, lo que hacíamos era esconder la estructura de la sucesión supercreciente eligiendo dos valores, M_0 (el módulo) y W_0 (el multiplicador), tal que $M_0 > \sum_{i=1}^{n} a'_i$, y que W_0 es primo

relativo con M_0, es decir, $\gcd(M_0, W_0) = 1$. Así, cada a'_i se transforma en un nuevo valor entre 0 y $M_0 - 1$ según la multipliación modular siguiente:

$$a_i \equiv W_0 \cdot a'_i \bmod M_0$$

La nueva sucesión $a = (a_1, ..., a_n)$ se publica como la clave pública.

Para demostrar que la complejidad asintótica del método criptoanalítico es polinómica, tenemos que considerar una familia de criptosistemas cuyo tamaño se incrementa indefinidamente. Hay dos parámetros básicos que tenemos que considerar: el número de elementos de la clave pública y su tamaño. Si estos se mantienen constantes, existe un algoritmo polinómico trivial para resolver los problemas de la mochila asociados. Por tanto, asumimos que el tamaño del módulo M_0 (y en consecuencia, también el tamaño de los elementos a_i) crece linealmente con n.

Si denominados d a la *constante de proporcionalidad* ($1 < d < \infty$), debemos escoger el valor a'_1 con $dn - n$ bits, a'_i con $dn - n + i - 1$ bits y M_0 con dn bits de tamaño. El parámetro d mide la redundancia introducida al criptosistema, es decir, la proporción entre los tamaños del texto cifrado y del texto plano. La complejidad de nuestro algoritmo es una función de d, que crece rápidamente, pero para cada valor fijo de d, es una función polinómica de n.

4.2. Descripción informal del algoritmo

El algoritmo propuesto por Shamir analiza los elementos de la clave pública $a = (a_1, ..., a_n)$ y trata de encontrar una pareja de naturales M y W que funcionen como valores trampilla, tal que $W \cdot a_i \bmod M$ sea una sucesión supercreciente y $M > \sum_{i=1}^{n} a_i$. Esto es, porque si se conoce cualquier par de números que cumplan con estas propiedades, entonces se pueden resolver eficientemente todos los problemas de la mochila relacionados con $a = (a_1, ..., a_n)$ en tiempo lineal.

Dado que los valores de a se obtienen a partir de una secuencia supercreciente mediante multiplicación modular, sabemos que al menos existe una pareja, que es la escogida por el diseñador. Ciertamente, el algoritmo encuentra una pareja de valores trampilla, pero no garantiza que se encuentre ni el módulo original, ni el multiplicador que se utilizó en la construcción de la clave pública.

Este algoritmo se divide en dos partes fundamentales. En la primera parte se utiliza el algoritmo de programación entera de Lenstra, para encontrar intervalos pequeños en $[0, 1]$ tal que siendo M y W una pareja de valores trampilla, entonces el ratio W/M quede dentro de dicho intervalo.

En la segunda parte del algoritmo, se usa el hecho de que conocemos el valor aproximado de W/M, para realizar un análisis más exhaustivo dividiendo cada intervalo en subintervalos más pequeños, de manera que si el ratio W/M queda dentro de un subintervalo,

entonces M y W componen una pareja de valores trampilla. Como ya hemos mencionado, al menos un subintervalo debe ser no vacío, ya que sabemos que existe una solución. Además, si utilizamos un algoritmo rápido de aproximación diofántica, encontraremos los valores más pequeños de M y W que verifican las condiciones.

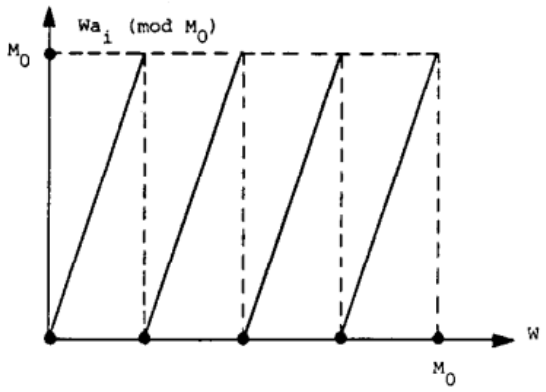

Figura 4.1: Gráfica de la función $W \cdot a_i \bmod M_0$, [22].

La figura 4.1 representa el gráfico de la función:

$$W \cdot a_i \bmod M_0 \text{ , con } 0 \leq W < M_0$$

Esta imagen tiene forma de dientes de sierra. En ella, estamos representando W en el eje horizontal y M_0 como el valor máximo del eje vertical, ya que es el módulo. La pendiente corresponde con el valor a_i, al igual que el número de mínimos; y la distancia entre dos mínimos sucesivos es igual a M_0/a_i, que es ligeramente superior a 1.

A continuación, vamos a resumir la idea principal para entender cómo se obtienen los valores M_0 y W_0. El valor W_0 verifica que $a'_1 = W_0 \cdot a_1 \bmod M_0$, es como máximo de tamaño 2^{dn-n}. Como la pendiente es a_1, entonces la distancia horizontal entre W_0 y el mínimo más cercano de la curva a_1, no puede ser mayor a $2^{dn-n}/a_1 \approx 2^{-n}$. Por tanto, la incógnita W_0 debe estar muy cerca de un mínimo de la curva a_1. Desafortunadamente, hay demasiados valores de W_0 y no podemos comprobar todos ellos.

Sin embargo, podemos aplicar un análisis análogo a la curva a_2, por lo que W_0 también debe estar a una distancia de $2^{dn-n+1}/a_2 \approx 2^{-n+1}$ del mínimo de a_2 más cercano. Consecutivamente, los mínimos de a_1 y a_2 deben estar muy cerca entre ellos. Esta condición de proximidad reduce el número de lugares en los que W_0 puede estar, aunque en la mayoría de casos sigue sin caracterizarlo de forma única.

De igual manera, si superponemos más de estas funciones sierra en un gráfico, todas las posibles ubicaciones de W_0 deben estar cerca de un mínimo en cada curva. Así que, en

lugar de encontrar un valor específico para W_0, debemos encontrar los puntos donde estos mínimos se agrupan, denominados *puntos de acumulación*.

Existe una sencilla regla empírica que podemos usar para tener una idea de cuántas de estas curvas debemos analizar al mismo tiempo, antes de que el conjunto de puntos de acumulación se reduzca a un tamaño manejable. Aunque esta estimación se ha probado en muchos experimentos y parece razonable, no siempre es totalmente precisa.

Supongamos que l representa la cantidad de curvas en forma de diente de sierra que superponemos en nuestro gráfico. Consideremos el p-ésimo mínimo de la curva a_1, que está situado en $W = pM_0/a_1$. Así, el mínimo más cercano de la curva a_i, se encuentra en algún lugar del intervalo:

$$[\frac{pM_0}{a_1} - \frac{M_0}{2a_i}, \frac{pM_0}{a_1} + \frac{M_0}{2a_i}]$$

cuya longitud es de $\frac{M_0}{a_i} \approx 1$. A continuación, hacemos una suposición bastante razonable, aunque nada rigurosa, de que la localización real de los distintos mínimos de a_i son variables aleatorias que siguen una distribución de probabilidad uniforme. Podemos así, estimar la probabilidad de que los mínimos de las curvas $a_2, ..., a_l$, estén lo suficientemente cerca del p-ésimo mínimo de la curva a_1, con la siguiente operación:

$$2^{-n+1} \cdot 2^{-n+2} \cdots 2^{-n+l-1} \approx 2^{-ln+n+l^2/2}$$

Puesto que debemos tener en cuenta también los a_1 posibles valores de p,

$$a_1 \cdot 2^{(-ln+n+l^2)/2} \approx 2^{dn-ln+n+l^2/2}$$

Este valor es menor que 1, cuando se verifica la siguiente desigualdad:

$$(l - d - 1)n > \frac{l^2}{2}$$

Por último, tomando n suficientemente grande, se verifica:

$$l > d + 1$$

De esta manera, l es una constante que depende de d, pero no de n.

La afirmación de que el número esperado de puntos de acumulación es menor que 1 no es literal, ya que siempre hay al menos un punto de acumulación "por construcción". No obstante, es razonable suponer que en situaciones prácticas, cuando l es mayor que $d + 1$, el punto "construido" no estará acompañado por otros puntos cercanos cuando se verifique la condición.

Ejemplo 4.1. [22] Por ejemplo, tomando $n = 100$ y $|M| = 200$, $l = 4$ puede ser un candidato razonable para indicar el número de curvas en forma de diente de sierra.

Sin embargo, todavía tenemos dos problemas: cómo deshacerse de M_0, cuyo valor aún es desconocido; y cómo encontrar los puntos de acumulación de los mínimos de las l curvas. Un punto a tener en cuenta es que los puntos de acumulación dependen de las pendientes de las curvas, pero no de su tamaño. Por tanto, si dividimos entre M_0 los dos ejes en la curva i-ésima, obtenemos la curva de la siguiente función:

$$V \cdot a_i \bmod 1 \text{ , con } 0 \leq V < 1$$

Siendo $V = W/M_0$. Además, esta curva es independiente de M_0, como se aprecia en la imagen:

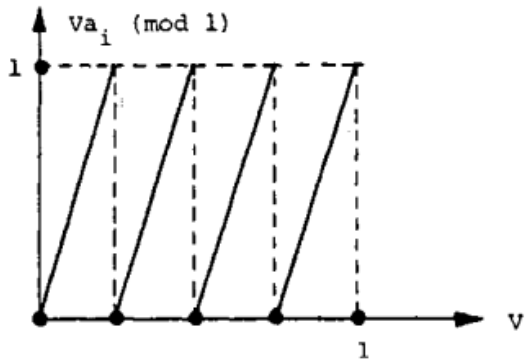

Figura 4.2: Gráfica de la función $V \cdot a_i \bmod 1$, [22].

En el nuevo sistemas de coordenadas, la pendiente de la curva sigue siendo a_i, al igual que el número de mínimos; pero la distancia entre mínimos sucesivos se reduce a $1/a_i$. Así, el parámetro W_0 queda reemplazado por el parámetro $V_0 = W_0/M_0$, y la distancia entre este parámetro y el mínimo de la curva a_i más cercano, se reduce aproximadamente en 2^{dn}. El problema de localización de los puntos de acumulación de l mínimos en el nuevo sistema de coordenadas, puede describirse mediante desigualdades lineales con l incógnitas integrales. Las condiciones para que el p-ésimo mínimo de a_1, q-ésimo mínimo de a_2, r-ésimo mínimo de a_3, ..., estén suficientemente próximos entre sí, son las siguientes:

$$\begin{aligned} &\text{Sean } p, q, r, ..., \text{ enteros,} && 1 \leq p \leq a_1 - 1 \\ &-\epsilon_2 \leq \frac{p}{a_1} - \frac{q}{a_2} \leq \epsilon_2' && 1 \leq q \leq a_2 - 1 \\ &-\epsilon_3 \leq \frac{p}{a_1} - \frac{r}{a_3} \leq \epsilon_3' && 1 \leq r \leq a_3 - 1 \\ &\vdots && \vdots \end{aligned}$$

donde ϵ_i y ϵ_i' representan las desviaciones hacia a la derecha y hacia la izquierda de p/a_1. multiplicando las inecuaciones dobles de la columna izquierda por sus denominadores,

obtenemos:

$$-\delta_2 \leq pa_2 - qa_1 \leq \delta_2'$$
$$-\delta_3 \leq pa_3 - ra_1 \leq \delta_3'$$
$$\vdots$$

Aquí se muestra que los valores de $a_2, a_3, ...$, se reducen simultáneamente a valores absolutos pequeños cuando se multiplica por p y se reduce mod a_1.

El problema de minimizar simultáneamente dos números mediante multiplicación modulo un tercer número, puede resolverse utilizando un sencillo algoritmo de fracciones continuas. En el caso general, debemos utilizar el algoritmo de programación entera de Lenstra, que aunque es más lento, es polinomial respecto al tamaño de los coeficientes para un número fijo de incógnitas. Este algoritmo es esencialmente un procedimiento de decisión que nos dice si un cierto sistema de desigualdades lineales tiene soluciones enteras. Al utilizar búsqueda binaria en los bits sucesivos de p, podemos encontrar todos los puntos de acumulación de las l curvas.

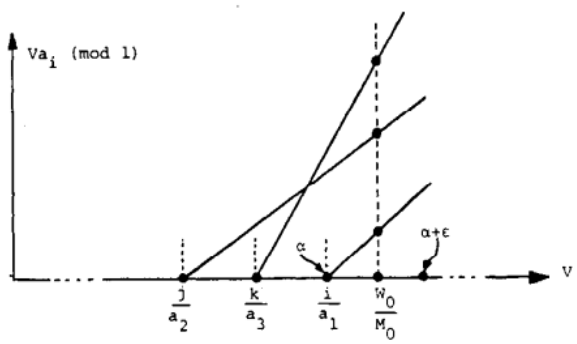

Figura 4.3: Sección ampliada del diagrama superpuesto cerca de $V_0 = \frac{W_0}{M_0}$, [22].

Para controlar el tiempo de ejecución del algoritmo y que sea polinomial, debemos añadir un parámetro k, el cual se encarga de abortar el programa si se supera cierto número k de puntos de acumulación (tomaremos $k = 100$ por defecto). Un ejemplo extremo de no utilizar este parámetro, sería que todos los a_i fueran iguales, por lo que todos los mínimos serían puntos de acumulación. Otra idea que podemos usar es cambiar el valor de l y de k, ya que así es posible controlar la fracción de claves para las que el algoritmo no consigue calcular una pareja de valores trampilla.

Observación 4.2. Cabe mencionar que no resolver todas las instancias de un problema no es una desventaja grave en el contexto de la criptografía, ya que un criptosistema se vuelve inútil cuando la mayoría de sus claves pueden ser criptoanalizados eficientemente, por lo que no hace falta que se resuelvan todas.

Podría suceder que un usuario, futuro receptor de un mensaje, permutase los elementos de su clave pública antes de mostrarla. Entonces, a_i ya no correspondería al i-ésimo elemento más pequeño de la sucesión supercreciente. Aún así, esta variante de Merkle-Hellman puede romperse en tiempo polinómico con esta técnica. Como sólo debemos buscar los l elementos más pequeños, estos se pueden encontrar en $O(n')$ maneras. Dado que l es una constante que no depende de la complejidad de n, nuestro método aumenta su complejidad en un factor polinomial.

Alternativamente, el criptoanalista puede relajar los valores de ϵ, que representan las desviaciones entre los distintos mínimos de la curva, de modo que el problema de programación entera pueda satisfacerse no sólo cuando se adivinen correctamente los l valores incrementales más pequeños, sino para cualquier elección de l valores lo suficientemente pequeños. Además, si se eligen adecuadamente los nuevos valores de los ϵ, es posible sustituir el factor $O(n')$ por una constante, lo que en aplicaciones prácticas ahorra tiempo.

El análisis de las primeras l curvas en diente de sierra, nos permite concentrarnos en unas pocas regiones pequeñas en las que debe localizarse el valor real de $V_0 = W_0/M_0$. Dentro de estas regiones, las curvas son lineales a trozos con sólo unos pocos puntos de discontinuidad, por lo que sus valores pueden expresarse y compararse sin un análisis excesivo de casos.

La segunda parte del algoritmo se encarga de descartar en estas regiones, todas aquellas subregiones en las que la sucesión de valores de la curva no sea supercreciente, o su suma sea menor que 1. Cada punto racional del resto de subregiones corresponde a una pareja de valores trampilla. Además como $V_0 = W_0/M_0$ no ha sido descartado, algún subconjunto es no vacío.

Sea ahora p uno de los valores obtenido por la primera parte del algoritmo. Consideramos el intervalo $[p/a_1, (p+1)/a_1)$ entre a_1 mínimos sucesivos. El número de puntos de discontinuidad esperados de otras curvas superpuestas en él, es $O(n)$. Sea $V_1, ..., V_s$ una lista de coordenadas de estos puntos discontinuos ordenadas de manera creciente. Entre cada V_t y V_{t+1}, todas las curvas a_i parecen segmentos lineales simples. El i-ésimo segmento lineal puede expresarse así:

$$Va_i - \tau_i^t \text{ , con } V_t \leq V < V_{t+1}$$

donde τ_i^t es el número de mínimos de a_i en $(0, V_t]$. Esto es, τ_i^t/a_i es el punto donde se corta la línea con el eje V. A continuación, podemos escribir el rango, tamaño y las condiciones de sucesión supercreciente de la siguiente forma:

$$V_t \leq V < V_{t+1}$$

$$\sum_{i=1}^{n} (Va_i - \tau_i^t) < 1$$

$$Va_i - \tau_i^t > \sum_{j=1}^{i-1} (Va_j - \tau_j^t) \text{ , con } i = 2, ..., n$$

La solución de este sistema de inecuaciones lineales en V es un subintervalo de la forma $[V_t, V_{t+1})$, posiblemente vacío. Además, se verifica que W/M pertenece a ese subintervalo para algún p y t, si y solo si, M y W forman una pareja de valores trampilla.

Solo falta mencionar que, en caso de que se realice una permutación de elementos de la sucesión supercreciente, es necesario usar también la condición de permutación supercreciente. Como no se puede averiguar la permutación aplicada en tiempo polinomial, debemos usar que toda sucesión supercreciente es una sucesión creciente, para reducir el número de permutaciones posibles a considerar.

Para ello, debemos ampliar la definición de la secuencia $V_1, ..., V_s$, incluyendo no solo los puntos de acumulación, sino también las V coordenadas de las intersecciones entre pares de curvas. Esto puede hacer que aumente el valor de s, pasando de $O(n)$ a $O(n^2)$.

Dentro de cada nuevo rango de valores $[V_t, V_{t+1})$, existe una forma clara de ordenar las diferentes curvas verticalmente. Por lo tanto, solo hay una forma posible de organizar sus nombres de manera que la secuencia sea creciente. En particular, de las $n!$ permutaciones posibles, sólo debemos considerar $O(n^2)$ en cada punto de acumulación.

4.3. Descripción formal del algoritmo

Como ya mencionamos anteriormente, el algoritmo abortará si las l curvas tienen al menos k puntos de acumulación. En esta sección vamos a analizar cómo la fracción de los parámetros l y k, influye en la cantidad de fallos del algoritmo y demostraremos que en un modelo probabilístico simplificado, esta fracción pueden hacerse arbitrariamente pequeña.

Por simplificar, asumimos que a_1 es un valor primo fijo y que $a_2, ..., a_l$ son variables aleatorias independientes que siguen una distribución de probabilidad uniforme en $[1, a_1 - 1]$. La condición de que a_1 sea primo es básicamente para que los inversos modulares estén bien definidos, pero no es esencial y por tanto, se puede reemplazar por otra condición o un análisis más cuidadoso. Por último, asumiremos por simplicidad que las variables del algoritmo de programación entera δ_i y δ_i' coinciden, y denotaremos ambas variables como δ, aunque esto sea un abuso de notación.

Definición 4.3. [22] Para cada $2 \leq i \leq l$, se define S_i como el conjunto de indices de mínimos de a_1 suficientemente cercanos a algún mínimo de a_i:

$$S_i = \{1 \leq p \leq a_1 - 1 \mid \exists\, q_i, 1 \leq q_i \leq a_i - 1, \text{ tq } -\delta \leq pa_i - qa_1 \leq \delta\}$$

Como todos los S_i son conjuntos de mínimos de a_1, su intersección $S_2 \cap ... \cap S_l$, es exactamente el conjunto de puntos de acumulación en los que un mínimo de a_1, está simultáneamente cerca de los mínimos de todas las demás curvas.

Lema 4.4. *[22] Mostramos ahora una caracterización alternativa al conjunto S_i definido previamente, más fácil de manipular:*

$$S_i = \{j_i a_i^{-1} \ mod \ a_1 \ | \ -\delta \leq j_i \leq \delta, j_i \neq 0\}$$

Demostración.

$$p \equiv j_i a_i^{-1} \bmod a_1$$

equivale a

$$pa_i \equiv j_i a_i^{-1} a_i \bmod a_1 \equiv j_i \bmod a_1$$

y por tanto, existe q_i tal que:

$$pa_i = j_i + q_i a_1$$

Como $-\delta \leq j_i \leq \delta$, $pa_i - q_i a_1$ cumple las condiciones. Finalmente, el valor $j_i = 0$ no está permitido en la definición de S_i, por lo que se restringe ese valor. □

La equivalencia $p \equiv j_i a_i^{-1} \bmod a_1$, establece para cada p, una relación uno a uno entre la secuencia $a_2, ..., a_l$ y la secuencia $j_2, ..., j_l$. El valor p es un punto de acumulación de $a_2, ..., a_l$, sí y solo si todos los correspondientes j_i son enteros no nulos pertenecientes a $[-\delta, \delta]$. De manera alternativa, cuando p y una secuencia de j_i son dados, existe una única secuencia de a_i para la cual p es un punto de acumulación con estos j_i índices.

Lema 4.5. *[22] Sean p' y p'' dos puntos de acumulación de $a_2, ..., a_l$, y sean $j'_2, ..., j'_l$ y $j''_2, ..., j''_l$ las secuencias de sus índices j asociados. Si $\delta < \sqrt{a_1/2}$, entonces ambas secuencias son múltiplos de una secuencia común $j_2, ..., j_l$ donde se verifica que gcd$(j_2, ..., j_l) = 1$.*

Demostración. Partiendo de $p' \equiv j'_i a_i^{-1} \bmod a_1$, y $p'' \equiv j''_i a_i^{-1} \bmod a_1$, podemos obtener:

$$a_i \equiv j'_i p'^{-1} \equiv j''_i p''^{-1} \bmod a_1$$

que, tras simplificación:

$$j'_i j_i''^{-1} \equiv p' p''^{-1} \bmod a_1$$

Como la parte derecha de la equivalencia no depende de i, para cualquier s y t se verifica:

$$j'_s j_s''^{-1} \equiv j'_t j_t''^{-1} \bmod a_1$$
$$j'_s j''_t \equiv j'_t j''_s \bmod a_1$$

Por la suposición de δ, cada producto $j'j''$ sólo puede estar en $[\frac{-a_1}{2}, \frac{a_1}{2}]$, por lo que se cumple (incluso sin módulo):

$$j'_s j''_t = j'_t j''_s$$

Esta igualdad se verifica para todo s y t sólo si las secuencias j' y j'' son múltiplos racionales entre sí. Como contienen sólo números enteros, deben ser múltiplos de alguna secuencia común $j_2, ..., j_l$ de enteros, cuyo gcd sea 1. □

Corolario 4.6. *Cuando $\delta < \sqrt{a_1/2}$ y la intersección $S_2 \cap ... \cap S_l$ es no vacía, hay un punto de acumulación cuya secuencia de índices es $j_2, ..., j_l$ y su gcd es 1, y además todos los puntos de acumulación restantes se obtienen multiplicando la secuencia de j_i por $1, -1, 2, -2, 3, -3, ...$ etc. hasta que un elemento supere el valor de δ.*

Cuando $\delta \geq \sqrt{a_1/2}$, la intersección $S_2 \cap ... \cap S_l$ es más complicada de analizar y no tenemos ninguna caracterización sencilla.

Definición 4.7. [22] Denotaremos como $N(l, k, \delta)$ al número de secuencias $a_2, ..., a_l$ en $[1, a_1 - 1]$, para las que la intersección $S_2 \cap ... \cap S_l$ contiene al menos k puntos cuando la distancia tolerable es δ.

Nos interesa la probabilidad condicional de que las l curvas tengan al menos k puntos de acumulación, cuando se sabe que tienen al menos uno. Dado que lo primero implica lo segundo, esta probabilidad condicional es simplemente:

$$\frac{N(l, k, \delta)}{N(l, 1, \delta)}$$

Lema 4.8. *[22] Para cada $\delta < \sqrt{a_1/2}$ y $l \geq 3$, hay una constante τ en $[3/\pi^2, 1/2]$ que depende solo de l, tal que:*

$$N(l, 1, \delta) \approx \tau(a_1 - 1)(2\delta)^{l-1}$$

Demostración. Podemos calcular el número de secuencias $a_2, ..., a_l$ que tienen al menos un punto de acumulación, contando el número de secuencias $p, a_2, ..., a_l$, donde p es un punto de acumulación de a_i. Este número es igual al número de secuencias $p, j_2, ..., j_l$, donde p es arbitrario y los j_i son enteros no nulos pertenecientes a $[-\delta, \delta]$, equivalentemente, $(a_1 - 1)(2\delta)^{l-1}$. Para corregir el cálculo, tomamos solamente secuencias de j_i cuyo gcd sea 1. Por el lema 4.5, para cada a_i, hay exactamente dos secuencias j_i con puntos de acumulación y con gcd = 1 (cada secuencia es la negación de la otra).

Para $l = 3$, la fracción de secuencias enteras de longitud $l - 1$ cuyo gcd = 1, converge a $6/\pi^2$ y para valores superiores de l esta fracción se aproxima a 1. Dado que cada secuencia a_i con puntos de acumulación se cuenta exactamente dos veces, tenemos que dividir esta constante por 2 para obtener la constante correcta τ. □

Lema 4.9. *[22] Si $\delta < \sqrt{a_1/2}$, entonces $N(l, k, \delta) \leq N(l, 1, \frac{\delta}{k/2})$.*

Demostración. Sea $j_2, ..., j_l$ la secuencia de índices con gcd = 1, cuya existencia está garantizada por el lema 4.5. Como $a_2, ..., a_l$ tiene al menos k puntos de acumulación, esta secuencia de j_i puede ser multiplicada por $k/2$ y todos sus elementos seguirán perteneciendo a $[-\delta, \delta]$. Como consecuencia, todos los j_i originales están en $[\frac{-\delta}{k/2}, \frac{\delta}{k/2}]$, y por tanto, la sucesión a_i tiene al menos un punto de acumulación incluso cuando el límite δ se sustituye por otro límite más estricto $\frac{\delta}{k/2}$. □

Llegamos a continuación al teorema principal de este capítulo.

Teorema 4.10. *[22] Cuando $\delta < \sqrt{a_1/2}$ y $l \geq 3$, el valor de la probabilidad condicionada $\frac{N(l,k,\delta)}{N(l,1,\delta)}$ es como máximo $(\frac{1}{k/2})^{l-1}$.*

Demostración.

$$\begin{aligned}\frac{N(l,k,\delta)}{N(l,1,\delta)} &\leq \frac{N(l,1,\delta/(k/2))}{N(l,1,\delta)} \\ &= \frac{\tau(a_1-1)(2\delta/(k/2))^{l-1}}{\tau(a_1-1)(2\delta)^{l-1}} \\ &= (\frac{1}{k/2})^{l-1}\end{aligned}$$

□

Ejemplo 4.11. [22] Tomando $l = 4$, $k = 100$ y $\delta < \sqrt{a_1/2}$, la probabilidad de que 4 curvas de dientes en sierra tengan al menos 100 puntos de acumulación cuando se sabe que tienen al menos uno, es de $(1/50)^3 = 1/125000$. Por tanto, si utilizamos el algoritmo de Lenstra (algoritmo L^3) para encontrar los puntos de acumulación y se aborta tras encontrar 100 puntos, la probabilidad de fallo es insignificante.

En nuestro análisis aplicado, δ es aproximadamente 2^{dn-n} y a_1 es aproximadamente 2^{dn}. Así, la condición $\delta < \sqrt{a_1/2}$ es equivalente a la condición $d < 2$. Shamir no pudo probar el limite superior del Teorema 4.10 para criptosistemas en los que la relación d, del tamaño del módulo entre el número de elementos, es mayor que 2. Sin embargo, Jeff Lagarias anunció posteriormente un límite superior diferente aplicable a todo el rango $1 < d < \infty$.

4.4. Análisis del algoritmo

En el apartado anterior se ha mostrado que todos los criptosistemas de una iteración de Merkle-Hellman pueden ser rotos en tiempo polinomial con probabilidad de fallo arbitrariamente pequeña. La parte que más tiempo necesita es la aplicación del algoritmo de programación entera de Lenstra, ya que en el peor caso su complejidad es polinomial en n, pero exponencial en l.

Una característica muy importante del ataque propuesto, es que se dirige a la clave pública y no a los textos cifrados individuales. De este modo, el criptoanalista puede abordar claves de reserva o de bajo volumen, incluso antes de que se utilicen por primera vez. Además, podría dedicar meses de tiempo computacional a cada clave, con el fin de facilitar posteriormente la decodificación del mensaje en microsegundos.

El problema más importante que queda abierto en este capítulo, es la seguridad criptográfica sobre los criptosistemas de Merkle-Hellman de varias iteraciones. En cada iteración, el módulo elegido aleatoriamente debe ser mayor que la suma de los elementos, por lo que las multiplicaciones modulares inversas reducen simultáneamente el tamaño de los elementos al menos en $log(n)$. En principio, esta condición permite hallar el único intervalo en el que se encuentra W/M, pero no W ni M.

En el caso de la mochila de iteración simple, cualquier pareja trampilla era útil, ya que generaba una secuencia supercreciente fácilmente resoluble. En el caso de las mochilas de iteración múltiple, sólo los valores W y M correctos permiten al criptoanalista hacer la multiplicación inversa correctamente y atacar las iteraciones internas una a una.

5 Ataques por baja densidad

El objetivo de este capítulo consiste en abordar otros ataques a criptosistemas distintos al que se examinó en el capítulo 4. En esta sección, nos enfocaremos en la vulnerabilidad de criptosistemas de baja densidad, como Merkle-Hellman iterado, cuya ruptura no fue tratada en el capítulo anterior. Las principales referencias bibliográficas de este capítulo son: [15], [13], [10] y [6].

5.1. Algoritmo L^3

Durante el transcurso de este capítulo, exploraremos diversos ataques criptográficos caracterizados por su preferencia a operar con criptosistemas de poca densidad. Ambos utilizan el algoritmo L^3, por lo que iniciaremos el capítulo proporcionando una explicación detallada de dicho algoritmo, para posteriormente adentrarnos en el análisis de cada uno de estos ataques.

El algoritmo L^3 o LLL, toma su nombre de sus diseñadores A. K. Lenstra, H. W. Lenstra, Jr. y L. Lovász. Es un algoritmo de simplificación de retículos publicado en 1982, que parte de una base con coordenadas n-dimensionales de un retículo y devuelve una base reducida del mismo retículo, en tiempo polinomial.

Aunque nos adentraremos en cada uno de los pasos de este algoritmo, primeramente debemos definir ciertos conceptos que nos serán necesarios.

Definición 5.1. [15] Sea el vector $v = (v_1, ..., v_n) \in \mathbb{R}^n$, definimos su *norma euclídea* o *longitud*, de la siguiente forma:

$$\|v\|^2 = \sum_{i=1}^{n} v_i^2$$

Llamaremos *vector corto* a un vector cuya longitud sea "pequeña". Buscamos encontrar el vector más corto, es decir, dada una base B hallar el $v \in \mathbb{B}$ tal que $\|v\| \leq \|x\|, \forall x \in B$.

Definición 5.2. [15] Un *retículo de enteros* L es un subgrupo aditivo de $\mathbb{Z}^n$, que contiene n vectores linealmente independientes sobre $\mathbb{R}^n$.

Definición 5.3. [15] Una *base* $(v_1, ..., v_n)$ de un retículo de enteros L es un conjunto de elementos de L que verifica:

$$L = \sum_{i=1}^{n} \mathbb{Z}v_i = \mathbb{Z}v_1 \oplus \cdots \oplus \mathbb{Z}v_n$$

Representamos una base del retículo L mediante la matriz de base n x n:

$$V = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$$

formada por los vectores de la base puestos por filas. Además, se verifica que si V_1 y V_2 son dos matrices de bases del mismo retículo L, existe una matriz unimodular $U \in GL(n, \mathbb{Z})$, que verifica que:

$$U \cdot V_1 = V_2$$

De igual forma, si V es una matriz de base del retículo L, y $U \in GL(n, \mathbb{Z})$, entonces $U \cdot V$ es una matriz de base de L.

Definición 5.4. [13] El *determinane* $d(L)$ de un retículo $L \subset \mathbb{R}^n$ se define por:

$$det(L) = |det(v_1, ..., v_n)|$$

donde los elementos v_i son los vectores de una base puestos por columnas. Este valor positivo no depende de la base escogida.

Ahora, sea $(v_1, ..., v_n)$ una base del retículo $L \subset \mathbb{R}^n$, y sea $(v'_1, ..., v'_n)$ su ortogonalización de Gram-Schmidt. Denotamos como (μ_{ij}) a los elementos de la matriz M de Gram-Schmidt que verifica $V = M \cdot V'$, donde V es la matriz con vector v_i en la fila i-ésima (respectivamente con V').

Definición 5.5. [15] Una base de un retículo $(v_1, ..., v_n)$ diremos que es *α-reducida* o *LLL-reducida con parámetro α*, si satisface ambas propiedades:

1. $|\mu_{ij}| \leq \frac{1}{2}$, para $1 \leq j < i \leq n$
2. $\|v'_i + \mu_{i,i-1}v'_{i-1}\|^2 \geq \alpha\|v'_{i-1}\|^2$, para $2 \leq i \leq n$

Esta última condición se conoce como *condición de intercambio*.

La idea intuitiva de la primera condición, nos dice que cada vector v_i de la base es "casi ortogonal" al generador de los vectores anteriores, ya que por Gram-Smith:

$$\langle v_1, ..., v_{i-1} \rangle = \langle v'_1, ..., v'_{i-1} \rangle$$

Por otro lado, la segunda condición nos indica que al intercambiar v_{i-1} y v_i, y aplicar de nuevo el método de ortogonalización de Gram-Schmidt, se puede generar un vector "un poco más corto".

Teniendo en cuenta que $\frac{1}{4} < \alpha < 1$, tomaremos el valor estándar del parámetro como $\alpha = \frac{3}{4}$. Nuestro objetivo es demostrar que para cualquier retículo $L \subset \mathbb{R}^n$ y cualquier base α-reducida de L, el algoritmo L^3 genera una base α-reducida de L en un número de pasos acotados por un polinomio dependiente de n.

Definición 5.6. [6] Definimos el parámetro secundario β como:

$$\beta = \frac{4}{4\alpha - 1} \text{ , con } \frac{1}{4} < \alpha < 1$$

Las dos condiciones de α son para que se verifique $\beta > \frac{4}{3}$. Asimismo, para el valor estándar $\alpha = \frac{3}{4}$, se obtiene $\beta = 2$.

Proposición 5.7. *[13] Sea $(v_1, ..., v_n)$ una base α-reducida de un retículo $L \subset \mathbb{R}^n$, y sea $(v'_1, ..., v'_n)$ su ortogonalización de Gram-Schmidt. Entonces:*

1. $|v_j|^2 \leq \beta^{i-j} |v'_i|^2$, *para* $1 \leq j \leq i \leq n$
2. $det(L) \leq |v_1| |v_2| \dots |v_n| \leq \beta^{n(n-1)/4} \cdot det(L)$
3. $|v_1| \leq \beta^{(n-1)/4} \cdot det(L)^{1/n}$

donde β es el parámetro definido previamente.

Demostración. Partiendo de la definición 5.5, podemos ver que:

$$|v'_i|^2 \geq (\alpha - \mu^2_{i,i-1}) |v'_{i-1}|^2 \geq (\alpha - \frac{1}{4}) |v'_{i-1}|^2 = \frac{1}{\beta} |v'_{i-1}|^2$$

Por tanto, $|v'_{i-1}|^2 \leq \beta |v'_i|^2$, y por inducción obtenemos:

$$|v'_j|^2 \leq \beta^{i-j} |v'_i|^2 \text{ , para } 1 \leq j \leq i \leq n \tag{5.1}$$

La definición de v'_i en la ortogonalización de Gram-Schmidt se puede reescribir como $v_i = v'_i + \sum_{j=1}^{i-1} \mu_{ij} v'_j$. Asimismo, como $v'_1, ..., v'_n$ son ortogonales, $v_i^2 = v_i'^2 + \sum_{j=1}^{i-1} \mu_{ij}^2 |v'_j|^2$. Con esto y a partir de la definición 5.5, llegamos a:

$$|v_i|^2 = |v'_i|^2 + \sum_{j=1}^{i-1} \frac{1}{4} \beta^{i-j} |v'_i|^2 = (1 + \frac{1}{4} \sum_{j=1}^{i-1} \beta^{i-j}) |v'_i|^2$$

Aplicando la fórmula de la suma de una progresión geométrica, obtenemos:

$$|v_i|^2 \leq (1 + \frac{1}{4} \cdot \frac{\beta^i - \beta}{\beta - 1}) \, |v_i'|^2$$

Ahora, vamos a aplicar inducción sobre i para demostrar lo siguiente:

$$1 + \frac{1}{4} \cdot \frac{\beta^i - \beta}{\beta - 1} \leq \beta^{i-1}$$

Para el primer caso, tomando $i = 1$ es trivial. Para el paso de inducción es suficiente probar que:

$$1 + \frac{1}{4} \cdot \frac{\beta^{i+1} - \beta}{\beta - 1} \leq \beta(1 + \frac{\beta^i - \beta}{4(\beta - 1)})$$

Como $\beta > \frac{4}{3}$, podemos multiplicar por $4(\beta - 1)$ y simplificar, obteniendo:

$$3\beta^2 - 7\beta + 4 = (\beta - 1)(3\beta - 4) \geq 0$$

Por tanto, ahora tenemos:

$$|v_i|^2 \leq \beta^{i-1} \, |v_i'|^2 \tag{5.2}$$

Usando la fórmula (5.1):

$$|v_j|^2 \leq \beta^{j-1} \, |v_j'|^2 \leq \beta^{i-1} \, |v_i'|^2 \text{, para } 1 \leq j \leq i \leq n$$

Por lo que queda demostrado el apartado (1). Por el teorema de Gram-Schmidt, sabemos que:

$$det(L) = |v_1'| \, |v_2'| \ldots |v_n'| \leq |v_1| \, |v_2| \ldots |v_n|$$

A partir de la ecuación (5.2),

$$|v_1|^2 \, |v_2|^2 \ldots |v_n|^2 \leq \beta^{0+1+\ldots+(n-1)} \, |v_1'|^2 \, |v_2'|^2 \ldots |v_n'|^2$$

y por tanto,

$$|v_1| \, |v_2| \ldots |v_n| \leq \beta^{n(n-1)/4} \, |v_1'| \, |v_2'| \ldots |v_n'| = \beta^{n(n-1)/4} \cdot det(L)$$

Por lo que queda demostrado el apartado (2). Por último, fijando $j = 1$ en el apartado (1), se tiene:

$$|v_1|^2 \leq \beta^{i-1} \, |v_i'|^2 \text{, con } 1 \leq i \leq n$$

Tomando el producto $i = 1, ..., n$:

$$|v_1|^{2n} \leq \beta^{0+1+\ldots+(n-1)} \, |v_1'|^2 \, |v_2'|^2 \ldots |v_n'|^2 = \beta^{n(n-1)/2} \cdot det(L)^2$$

Finalmente, tras tomar la raiz de $2n$, queda demostrado el apartado (3). □

Teorema 5.8. *[13] Sea $(v_1, ..., v_n)$ una base α-reducida del retículo $L \subset \mathbb{R}^n$, e $y \in L$ un vector no nulo del retículo, entonces:*

$$|v_1|^2 \leq \beta^{n-1} |y|^2$$

Demostración. Sea $(v'_1, ..., v'_n)$ la ortogonalización de Gram-Schmidt de $(v_1, ..., v_n)$. Por la definición de base α-reducida, para $2 \leq i \leq n$, tenemos:

$$|v'_i|^2 \geq (\alpha - \mu^2_{i,i-1}) |v'_{i-1}|^2 \geq (\alpha - \frac{1}{4}) |v'_{i-1}|^2 = \frac{1}{\beta} |v'_{i-1}|^2$$

Como $v_1 = v'_1$,

$$|v_1|^2 = |v'_1|^2 \leq \beta |v'_2|^2 \leq \beta^2 |v'_3|^2 \leq \cdots \leq \beta^{n-1} |v'_n|^2$$

por lo que tenemos:

$$|v'_i|^2 \geq \beta^{-(i-1)} |v_1|^2 \text{, para } 1 \leq i \leq n$$

Como consecuencia de Gram-Schmidt, se sabe que para cualquier $y \in L$ no nulo:

$$|y| \geq min\{|v'_1|, ..., |v'_n|\}$$

por tanto,

$$|y|^2 \geq min\{|v'_1|^2, ..., |v'_n|^2\} \geq \beta^{-(n-1)} |v_1|^2$$

Finalmente, despejando llegamos al resultado deseado:

$$|v_1|^2 \leq \beta^{n-1} |y|^2$$

□

Cabe destacar que el resultado anterior es el caso un particular, con $m = 1$, del siguiente resultado más general, cuya demostración obviaremos.

Teorema 5.9. *[13] Sea $(v_1, ..., v_n)$ una base α-reducida del retículo $L \subset \mathbb{R}^n$, e $(y_1, ..., y_m) \in L$, m vectores linealmente independientes del retículo, entonces, para $1 \leq j \leq m$ se verifica:*

$$|v_j|^2 \leq \beta^{n-1} max\{|y_1|^2, ..., |y_m|^2\}$$

Tras todo lo explicado, llegamos al objetivo principal de este primer apartado: el Algoritmo L^3. Para ello, incluimos el código [6] tanto del algoritmo principal, como de dos funciones utilizadas en él:

Algorithm 6 Función reduce(k, l)

Input: Índices de los valores que se van a "ortogonalizar".
Output: Base y coeficientes de la ortogonalización de Gram-Schmidt tras la modificación.

1: **if** $|\mu_{kl}| > 1/2$ **then**
2: $\quad y_k = y_k - [\mu_{kl}]y_l$
3: $\quad$ **for** j from 1 to $l-1$ **do**
4: $\quad\quad \mu_{kj} = \mu_{kj} - [\mu_{kl}]\mu_{lj}$
5: $\quad$ **end for**
6: $\quad \mu_{kl} = \mu_{kl} - [\mu_{kl}]$
7: **end if**

Esta primera función, hace que y_k sea "casi ortogonal" a y_l. Como $[\mu_{kl}]$ es el entero más cercano al coeficiente de Gram-Schmidt, es la mejor reducción posible. Después, actualiza la base y los coeficientes de la ortogonalización de Gram-Schmidt.

Algorithm 7 Función exchange(k)

Input: Índice del valor que se va a intercambiar con su anterior.
Output: Base y coeficientes de la ortogonalización de Gram-Schmidt tras el intercambio.

1: $z = y_{k-1}$; $y_{k-1} = y_k$; $y_k = z$
2: $v = \mu_{k,k+1}$; $\delta = \gamma'_k + v^2 \cdot \gamma'_{k-1}$
3: $\mu_{k,k-1} = v \cdot \gamma'_{k-1}/\delta$; $\gamma'_k = \gamma'_k\gamma'_{k-1}/\delta$; $\gamma'_{k-1} = \delta$
4: **for** j from 1 to $k-2$ **do**
5: $\quad t = \mu_{k-1,j}$; $\mu_{k-1,j} = \mu_{kj}$; $\mu_{kj} = t$
6: **end for**
7: **for** i from $k+1$ to n **do**
8: $\quad \psi = \mu_{ik}$; $\mu_{ik} = \mu_{i,k-1} - v \cdot \mu_{ik}$; $\mu_{i,k-1} = \mu_{k,k-1}\mu_{ik} + \psi$
9: **end for**

Esta otra función, intercambia los vectores y_{k-1} e y_k. Luego, actualiza la base y los coeficientes de la ortogonalización de Gram-Schmidt, y los devuelve.

Algorithm 8 Algoritmo L^3

Input: Una base $(x_1, ..., x_n)$ del retículo $L \subset \mathbb{R}^n$ y el parámetro $\alpha \in \mathbb{R}$ verificando $\frac{1}{4} < \alpha < 1$.
Output: Una base α-reducida $(y_1, ..., y_n)$ del retículo $L \subset \mathbb{R}^n$.

for i from 1 to n **do**
  $y_i = x_i$
end for
for i from 1 to n **do**
  $y'_i = y_i$
  for j from 1 to $i - 1$ **do**
    $\mu_{ij} = (y_i \cdot y'_j)/\gamma'_j$
    $y'_i = y'_i - \mu_{ij} \cdot y'_j$
  end for
  $\gamma'_i = y'_i \cdot y'_i$
end for
$k = 2$
while $k \leq n$ **do**
  reduce$(k, k-1)$
  if $\gamma'_k \geq (\alpha - \mu^2_{k,k-1})\gamma'_{k-1}$ **then**
    for l from $k-2$ to 1 **do**
      reduce(k, l)
    end for
    $k = k + 1$
  else
    exchange(k)
    if $k > 2$ **then**
      $k = k - 1$
    end if
  end if
end while

Por último, el algoritmo LLL realiza una copia de los vectores de entrada, aplica el proceso de ortogonalización de Gram-Schmidt y finalmente llama a las dos funciones previas para que reduzcan e intercambien los vectores obtenidos. Estos vectores se modifican continuamente a lo largo del algoritmo, de tal manera que siempre forman parte de una base del retículo L.

5.2. Ataque de Lagarias-Odlyzko

En esta sección se presenta el algoritmo diseñado por J. C. Lagarias y A. M. Odlyzko [15], para resolver instancias del problema de la mochila. Su método implica transformar la instancia original del problema y luego aplicar el algoritmo de reducción de bases de retículos, para así determinar la solución. El rendimiento del algoritmo propuesto se analiza detenidamente, y aunque este algoritmo siempre se detiene en tiempo polinómico, no siempre encuentra una solución cuando existe.

Para caracterizar el algoritmo, usaremos el concepto de densidad, visto en la definición 3.8, donde n representa la cantidad de elementos del conjunto. Así, para "casi todos" los casos con densidad $d < 0.645$, el vector que estamos tratando de encontrar resulta ser el vector más corto no nulo del retículo. Asimismo, tras realizar exhaustivas pruebas computacionales, se observa que este algoritmo es efectivo para densidades $d < d_c(n)$, donde $d_c(n)$ es un umbral. Como consecuencia, este enfoque proporciona una solución de tiempo polinómico a los sistemas de cifrado del problema de la mochila, lo que implica que puede comprometerlos si la velocidad de transmisión de información es menor que de $d_c(n)$, cuando $n \to \infty$.

5.2.1. Preludio al método

Tras la publicación del criptoanálisis de Shamir en 1983, que como ya sabemos, anunciaba un método para romper el criptosistema de clave pública más simple de Merkle-Hellman, se propusieron otros ataques para criptosistemas más complicados. Esos ataques estaban todos basados en la idea de recuperar la información trampilla oculta en los valores de los pesos. Sin embargo, aquí se propone un método denominado "Algorithm SV" (Short Vector), que localiza la solución del problema de la mochila directamente, sin intentar buscar la información trampilla.

Este nuevo método consiste en transformar el problema inicial, en el problema de encontrar el vector más corto e en un retículo de enteros L. Tras esto, se aplica el algoritmo previo de reducción de bases de retículos para obtener una base reducida. El método se considera exitoso cuando el vector e se encuentra en la base reducida, y en tal caso, podemos derivar una solución a partir del vector obtenido.

Recordemos que al explicar el problema de la mochila por primera vez, definíamos $a = \{a_1, ..., a_n\} \subseteq \mathbb{N}^*$ como el conjunto de pesos del problema, y $S \in \mathbb{N}^*$ la capacidad total de la mochila. Este recordatorio de la notación es importante para lo que explicaremos a continuación.

El principal resultado de esta sección se centra en el análisis del rendimiento del Algoritmo SV, dado que se muestra exitoso en la resolución de problemas de suma de subconjuntos de baja densidad de la siguiente forma:

1. Para "casi todos" los problemas de la mochila cuya $d(a) < 0.645$, el vector e es el vector no nulo más corto del retículo L.
2. Para "casi todos" los problemas de la mochila resolubles con n pesos que tienen $d(a) < (2-\epsilon)(log_2(\frac{4}{3}))^{-1}n^{-1}$, para cualquier $\epsilon > 0$ fijo, el algoritmo encuentra una solución.

Lagarias y Odlyzko explican en su artículo que el primer resultado es el más exacto posible, en el sentido de que ya no se cumple si se reemplaza por 0.646, según las heurísticas utilizadas. Por otro lado, el segundo resultado es más débil de lo que parece a simple vista, ya que no se garantiza que el algoritmo L^3 produzca el vector más corto no nulo v_{min} del retículo $L \subset \mathbb{Z}^n$, simplemente produce uno de ellos.

El método que presentamos utiliza el algoritmo L^3, ya que es el algoritmo original del artículo [15]. Además, funciona de manera satisfactoria para encontrar vectores razonablemente cortos de un retículo en tiempo de ejecución polinómico. Sin embargo, en lugar de usar el Algoritmo SV, se podrían realizar modificaciones sustituyendo el algoritmo L^3 por otro que encuentre vectores cortos de un retículo, o incluso, utilizar otro para encontrar buenas aproximaciones diofánticas multidimensionales, que podría funcionar bien en la práctica.

Para finalizar esta sección, debemos remarcar que este método complementa los ataques existentes a los criptosistemas de mochila que se basan en la recuperación de información trampilla. De este modo, cuando la densidad es baja, el método descrito aquí debería tener éxito. En cambio, cuando la densidad es alta, la información trampilla es más difícil de ocultar, y los ataques basados en esta búsqueda tienen más posibilidades de éxito.

5.2.2. Descripción del método

Procedemos en este apartado a explicar el método propuesto por Lagarias y Odlyzko para romper criptosistemas con poca densidad basados en el problema de la mochila. Tal y como ya hemos explicado, el objetivo es aplicar una reducción del problema de la mochila al problema de la búsqueda del vector más corto de un retículo, para así poder aplicar el algoritmo L^3, obteniendo la solución deseada.

Si el Algoritmo SV produce una solución, decimos que tiene éxito; si no, que falla. A continuación, se muestran los pasos del algoritmo a seguir:

Algorithm 9 Algoritmo Short Vector (SV)

Input: Clave pública $a = (a_1, ..., a_n)$ y capacidad máxima de la mochila M.
Output: Una solución $b'_i = (b'_{i,1}, ..., b'_{i,n+1})$ al problema de la mochila.

1: Considerar los siguientes vectores que forman una base $(b_1, ..., b_{n+1})$ de un retículo de enteros L con dimensión $n + 1$:

$$\begin{aligned} b_1 &= (1, 0, ..., 0, -a_1) \\ b_2 &= (0, 1, ..., 0, -a_2) \\ &\vdots \\ b_n &= (0, 0, ..., 1, -a_n) \\ b_{n+1} &= (0, 0, ..., 0, M) \end{aligned}$$

2: Aplicar el algoritmo L^3 para encontrar una base reducida $(b'_1, ..., b'_{n+1})$.
3: Comprobar si algún $b'_i = (b'_{i,1}, ..., b'_{i,n+1})$ tiene todos sus elementos $b'_{ij} = 0$ o $b'_{ij} = \lambda$, para algún λ fijo, con $1 \leq j \leq n$. Para cada b'_i que verifique lo anterior, comprobar si $x_j = \lambda^{-1} b'_{i,j}$ para $1 \leq j \leq n$, es solución. En caso afirmativo, finalizar el algoritmo. De lo contrario, continuar.
4: Repetir los pasos $(1) - (3)$ reemplazando M por $M' = \sum_{i=1}^{n} a_i - M$. Después, finalizar el algoritmo.

Como este algoritmo consiste esencialmente en aplicar dos veces el algoritmo L^3, obtenemos el siguiente lema acerca del límite en tiempo de ejecución. Obviaremos su demostración.

Lema 5.10. *[15] Sea $a = (a_1, ..., a_n)$ y $M < \sum_{i=1}^{n} a_i$ las entrada del algoritmo SV, y supongamos que $max(a_i) \leq B$. Entonces el Algoritmo SV se detiene tras, como mucho, $O(n^6(log(nB))^3)$ operaciones de bits.*

5.3. Ataque de Coster et al

De manera análoga al apartado anterior, vamos a explicar el algoritmo propuesto por Coster [10], que resuelve el problema de la mochila mejorando la solución de Lagarias recién vista. Realmente, el artículo original está escrito por los siguientes seis autores: Matthijs J. Coster, Antoine Joux, Brian A. Lamacchia, Andrew M. Odlyzko, Claus-Peter Schnorr y Jacques Stern. Aún así, usaremos el apellido de Coster para referirnos a todos los autores involucrados en este trabajo.

El resultado principal de este artículo es la obtención de una mejora al "Algoritmo SV" de Lagarias y Odlyzko, utilizando la misma idea de reducción del problema de la mochila al

problema de búsqueda del vector más corto en un retículo. Hasta ese momento, el mejor algoritmo para solventar este problema era el algoritmo L^3.

Sabemos que para poder encontrar una solución al problema de la mochila utilizando el Algoritmo SV de Lagarias-Odlyzko, la densidad de la clave pública debe ser menor a 0.645. Sin embargo, según la explicación de Coster, es posible hallar soluciones en tiempo polinomial para casi todos los problemas de suma de subconjuntos con densidad menor a 0.9408... mediante unas pequeñas modificaciones. A continuación, explicamos esta idea.

5.3.1. Preludio al método

En primer lugar, Coster divide su artículo en cinco secciones en las que trata diversos temas. La primera sección se centra en presentar el problema de la mochila, introducir los criptosistemas basados en este problema y resumir el artículo de Lagarias. En contraste, la última sección se encarga de analizar los resultados obtenidos.

Es en las secciones intermedias, donde se propone una modificación del Algoritmo SV, que irá mejorando con el avance del artículo. Detallamos ahora estas secciones, con el fin de comprender el proceso de perfeccionamiento que lleva a la obtención de los algoritmos finales.

5.3.1.1. Segunda sección

Comencemos definiendo la clave pública como $(a_1, ..., a_n)$ y el vector solución del problema de la mochila como $e = (e_1, ..., e_n) \in \{0, 1\}^n$. Así, $s = \sum_{i=1}^{n} e_i a_i$, es la capacidad máxima de la mochila, es decir, el mensaje cifrado; y $t = \sum_{i=1}^{n} a_i$, es la suma de la clave pública.

Vamos a asumir que $s \geq t/n$, ya que de lo contrario si $s < t/n$, nigún $a_i \geq t/n$ puede estar en el subconjunto. De forma similar, $s \leq (1 - (1/n))t$ ya que sino cualquier $a_i > t/n$ debe estar en el subconjunto. Obtenemos por tanto:

$$\frac{1}{n}t \leq s \leq \frac{n-1}{n}t$$

Seguidamente, vamos a redefinir el ataque de Lagarias-Odlyzko a criptosistemas de baja densidad basados en el problema de la suma de subconjuntos. Definimos así $b_1, ..., b_{n+1}$, como los siguientes vectores:

$$\begin{aligned} b_1 &= (1, 0, ..., 0, Na_1) \\ b_2 &= (0, 1, ..., 0, Na_2) \\ &\vdots \\ b_n &= (0, 0, ..., 1, Na_n) \\ b_{n+1} &= (0, 0, ..., 0, Ns) \end{aligned}$$

donde N es un entero positivo que verifica $N > \sqrt{n}$.

Para saber de donde sale este valor, definamos L como el retículo generado por los vectores anteriores $b_1, ..., b_{n+1}$:

$$L = \{\sum_{i=1}^{n+1} z_i b_i, \text{ tq } z_i \in \mathbb{Z} \text{ para } 1 \leq i \leq n+1\}$$

Podemos ver que el vector $\hat{e} = (e_1, ..., e_n, 0) \in L$. Por temas de demostración, estamos interesados en los vectores $\hat{x} = (x_1, ..., x_{n+1})$ que verifican:

$$\begin{aligned} &\|\hat{x}\| \leq \|\hat{e}\|, \\ &\hat{x} \in L, \\ &\hat{x} \notin \{0, \hat{e}, -\hat{e}\} \end{aligned} \tag{5.3}$$

Además, supongamos que:

$$\sum_{i=1}^{n} e_i \leq \frac{1}{2} n$$

Esto nos dice que el subconjunto contiene como máximo la mitad de los a_i totales. Si contrariamente $\sum_{i=1}^{n} e_i > \frac{1}{2} n$, debemos reemplazar s por $t - s$, b_{n+1} por $b'_{n+1} = (0, ..., 0, N(t-s))$ y $\hat{e}$ por $\hat{e}' = (1 - e_1, ..., 1 - e_n, 0)$. Resolver este nuevo problema es equivalente a resolver el problema $\sum_{i=1}^{n} (1 - e_i) \leq \frac{1}{2} n$ y $s' = t - s \geq t/n$.

Por último, tomando $N > \sqrt{n}$, es claro que $\hat{x}$ verifica la ecuación (5.3) solo si $x_{n+1} = 0$, ya que de otro modo contradice a la ecuación:

$$\|\hat{x}\| \geq |x_{n+1}| \geq N > \sqrt{n} \geq \|\hat{e}\|$$

El resto de la sección se ocupa de demostrar que todos los problemas de suma de subconjuntos con densidad menor a 0.6463... podrían resolverse en tiempo polinómico, dada la existencia de un oráculo.

Definición 5.11. Definimos un *oráculo de retículo*, o simplemente *oráculo*, a una caja negra o algoritmo teórico que es capaz de obtener el vector más corto de un retículo en tiempo polinomial.

5.3.1.2. Tercera sección

La tercera sección del artículo introduce una nueva mejora de la densidad máxima de los problemas de suma de subconjuntos que pueden resolverse "casi siempre":

Teorema 5.12. *[10] Sea A un número entero positivo, y sean $a_1, ..., a_n$ enteros aleatorios con $0 < a_i \leq A$ para $1 \leq i \leq n$. Sea $e = (e_1, ..., e_n) \in \{0,1\}^n$ arbitrario, y $s = \sum_{i=1}^{n} e_i a_i$. Si la densidad $d < 0.9408...$, entonces el problema de la suma de subconjuntos definido por $a_1, ..., a_n$ y s, puede resolverse "casi siempre" en tiempo polinómico con una sola llamada a un oráculo.*

Para obtener este resultado, debemos reemplazar el vector anterior b_{n+1}, por:

$$b'_{n+1} = (\frac{1}{2}, ..., \frac{1}{2}, Ns)$$

donde ahora, N es un entero positivo que verifica $N > \frac{1}{2}\sqrt{n}$.

Para explicar este nuevo valor, definamos el retículo L' como el generado por los vectores $b_1, ..., b'_{n+1}$. Nos damos cuenta de que el vector $\hat{e} \notin L'$. No obstante, si que se verifica que $\hat{e}' = (e'_1, ..., e'_n, 0) \in L'$, con $e'_i = e_i - \frac{1}{2}$. Como $e_i \in \{0, 1\}$ para $1 \leq i \leq n$, sabemos que $e'_i \in \{-\frac{1}{2}, \frac{1}{2}\}$ para $1 \leq i \leq n$. Notemos también que $\|\hat{e}'\|^2 \leq \frac{1}{4}n$.

Finalmente, aplicando el mismo argumento previo y tomando $N > \frac{1}{2}\sqrt{n}$, es claro que $\hat{x}$ verifica la ecuación (5.3) solo si $x_{n+1} = 0$, de ahí su valor. El resto de la tercera sección se enfoca en demostrar que gracias a esta modificación, cualquier problema de suma de subconjuntos con densidad $d < 0.9408...$, puede ser resuelto en tiempo polinómico, dada la existencia de un oráculo.

5.3.1.3. Cuarta sección

En la tercera sección, hemos presentado una manera de mejorar la densidad, por debajo de la cual un oráculo permite resolver la mayoría de los problemas de suma de subconjuntos. Esto lo hemos hecho reemplazando el retículo L, por el retículo L'. Aquí esbozamos cómo se puede lograr un aumento en la densidad crítica comparable a este último apartado, mediante el uso del retículo L'', que es muy diferente.

Este nuevo retículo L'' se genera mediante los siguientes vectores en $\mathbb{R}^{n+2}$:

$$\begin{aligned}
b_1 &= (n+1, -1, -1, ..., -1, Na_1) \\
b_2 &= (-1, n+1, -1, ..., -1, Na_2) \\
&\vdots \\
b_n &= (-1, ..., -1, n+1, -1, Na_n) \\
b_{n+1} &= (-1, ..., -1, -1, n+1, -Ns)
\end{aligned}$$

donde N es un entero positivo que verifica $N \geq n^2$.

Durante el resto de esta sección, se prueba que la densidad crítica de este método es exactamente la misma que la del método de la sección 3, ya que ambos dependen del número de puntos del retículo en cualquier esfera (en $\mathbb{R}^n$, para el método de la sección 3; o $\mathbb{R}^{n+1}$, para el método de esta sección) que tenga un radio aproximado de $\frac{1}{2}\sqrt{n}$ menor que A. Sin embargo, el retículo L'' utilizado en esta sección es muy diferente del retículo L' de la sección anterior.

La razón principal por la cual el retículo L tiene peor rendimiento que los retículos L' y L'', es que contiene muchos vectores cortos en los cuales algunas de las primeras n

coordenadas son -1. Concluimos de esta forma con la explicación progresiva del artículo de Coster, y procedemos a la descripción del método basado en el retículo L'.

5.3.2. Descripción del método

En este apartado vamos a describir el método de Coster para romper criptosistemas basados en el problema de la mochila. Como ya sabemos, al igual que en el método de Lagarias-Odlyzko, el objetivo es aplicar una reducción del problema de la mochila al problema de la búsqueda del vector más corto de un retículo, para así poder aplicar el algoritmo L^3, obteniendo la solución deseada. La diferencia con el método anterior radica en la matriz inicial escogida. A continuación, se muestran los pasos a seguir en este algoritmo mejorado, explicado en la tercera sección del artículo de Coster.

Algorithm 10 Algoritmo Coster

Input: Clave pública $a = (a_1, ..., a_n)$ y capacidad máxima de la mochila s.
Output: Una solución $b'_i = (b'_{i,1}, ..., b'_{i,n+1})$ al problema de la mochila.

1: Considerar los siguientes vectores que forman una base $(b_1, ..., b_{n+1})$ de un retículo de enteros L' con dimensión $n+1$:

$$\begin{aligned} b_1 &= (1, 0, ..., 0, Na_1) \\ b_2 &= (0, 1, ..., 0, Na_2) \\ &\vdots \\ b_n &= (0, 0, ..., 1, Na_n) \\ b_{n+1} &= (\frac{1}{2}, ..., \frac{1}{2}, Ns) \end{aligned}$$

donde N es un valor verificando $N > \frac{1}{2}\sqrt{n}$.
2: Aplicar el algoritmo L^3 para encontrar una base reducida $(b'_1, ..., b'_{n+1})$.
3: Comprobar si algún $b'_i = (b'_{i,1}, ..., b'_{i,n+1})$ es solución. En caso afirmativo, finalizar el algoritmo. De lo contrario, continuar.
4: Intercambiar los valores $\frac{1}{2}$ por $-\frac{1}{2}$ y viceversa, de la matriz obtenida tras el apartado (2), y volver a aplicar el paso (3). Tras esto, finalizar el algoritmo.

Tras todo el análisis realizado, sabemos que es posible mejorar el límite de densidad de 0.6463... a 0.9408..., modificando un vector de la base del retículo. Consideremos ahora las posibilidad de superar este límite.

Realmente, resolver problemas de suma de subconjuntos con reducción de base está estrechamente relacionado con problemas de cobertura de retículos. En particular, queremos cubrir los vértices del n-cubo (que representa los posibles vectores solución) con un número polinómico de n-esferas de radio $\sqrt{\alpha n}$.

Lagarias y Odlyzko demostraron que era posible cubrir el n-cubo con dos n-esferas de radio $\sqrt{\frac{1}{2}n}$, centradas en $(0,0,...,0)$ y $(1,1,...,1)$. Estas dos n-esferas corresponden a los dos problemas de reducción de base que deben resolverse para cualquier problema dado de suma de subconjuntos. Así, el análisis de Coster, utiliza una n-esfera de radio $\frac{1}{2}\sqrt{n}$ centrada en $(\frac{1}{2},\frac{1}{2},...,\frac{1}{2})$, para cubrir todos los puntos.

Una forma de mejorar el límite presentado anteriormente, sería demostrar que es posible cubrir los vértices del n-cubo con un número polinómico de n-esferas de radio $\sqrt{\alpha n}$, con $\alpha < \frac{1}{4}$. Mostramos a continuación que esto no es posible y que el límite asintótico de 0.9408... no puede ser mejorado de esta manera.

La siguiente proposición (cuya demostración obviaremos) muestra que cualquier n-esfera de radio $\sqrt{\alpha n}$ con $\alpha < \frac{1}{4}$, puede solamente cubrir una fracción exponencialmente pequeña de los vértices del n-cubo. Por lo tanto, ninguna colección polinómica de dichas esferas puede satisfacer nuestros requisitos.

Proposición 5.13. *[10] Cualquier esfera de radio $\sqrt{\alpha n}$ con $\alpha < \frac{1}{4}$ en $\mathbb{R}^n$, contiene como máximo $(2-\delta)^n$ puntos en $\{0,1\}^n$, para $\delta = \delta(\alpha) > 0$.*

A medida que n tiende a infinito, cualquier n-esfera de radio $\sqrt{\alpha n}$ con $\alpha < \frac{1}{4}$, contendrá como máximo $(2-\delta(\alpha))^n$ puntos en $\{0,1\}^n$. Por tanto, cualquier colección de esferas de tamaño polinómico no puede contener todos los puntos en $\{0,1\}^n$. Como consecuencia, no podemos mejorar el límite asintótico de 0.9408... al reducir un número polinómico de bases con b_{n+1} vectores distintos. Sin embargo, para dimensiones pequeñas, podría ser posible mejorar el límite, aunque cualquier ventaja de este tipo desaparecerá a medida que n aumente.

5.4. Resultados Experimentales

A continuación, vamos a analizar los datos obtenidos en un experimento llevado a cabo con el fin de evaluar la relación existente entre el tamaño del mensaje y el resultado del ataque de Coster. Para ello, creamos una función (medirErrores) en la implementación del método de Coster (Coster.ipynb), que obtiene n valores de densidad en el intervalo $(0,1)$ y genera 10 criptosistemas de Merkle-Hellman para cada valor.

De esta forma, si el método de Coster consigue obtener el resultado de Merkle-Hellman para alguno de esos 10 criptosistemas, marcamos ese punto de la gráfica en color verde. Si por el contrario no lo consigue para ninguno de los 10 criptosistemas, marcamos ese punto de color rojo. Así obtenemos la siguiente gráfica, donde el eje de abcisas muestra el tamaño de mensaje y el eje de ordenadas muestra la densidad de la clave pública.

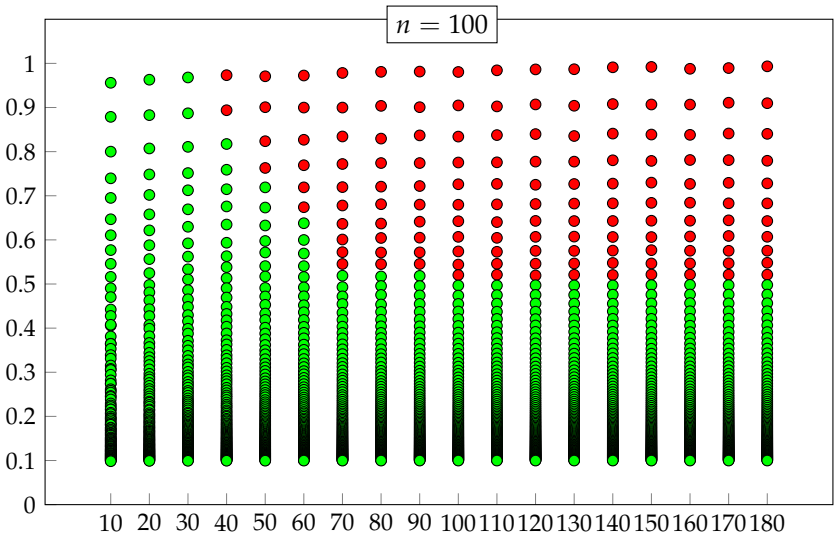

Figura 5.1: Relación tamaño-densidad

A simple vista, podemos observar cómo para tamaños menores a 40, el método de Coster consigue obtener todos los mensajes cifrados con Merkle-Hellman, incluso aquellos con densidades cercanas a 1. Sin embargo, entre los tamaños 40 y 70, se evidencia que el método requiere densidades más bajas para poder obtener una solución.

Finalmente, es claro que a partir de tamaño 70, el gráfico se estabiliza en una densidad cercana a 0.52. No obstante, es interesante saber que, aún no siendo tan apreciable, el metódo sigue necesitando reducir esa densidad hasta 0.497 para recuperar el mensaje de Merkle-Hellman de tamaño 180.

6 Criptosistema de Chor-Rivest

En el presente capítulo, dirigiremos nuestra atención hacia el criptosistema de Chor-Rivest. Estudiaremos este sistema de cifrado, veremos su implementación y analizaremos su robustez ante los ataques vistos. Para el desarrollo de este capítulo, se han utilizado fundamentalmente las siguientes bibliografías: [5] y [9].

6.1. Introducción

Como ya sabemos, en 1976, Diffie y Hellman introdujeron la idea de los criptosistemas de clave pública (PKC's), para los cuales se utilizan dos claves diferentes: una para cifrar y otra para descifrar. Además, conocemos el objetivo de ambas claves: cada usuario mantiene en secreto su clave de descifrado al mismo tiempo que hace pública su clave de cifrado, de modo que pueda ser utilizada por cualquier persona que desee enviarle un mensaje.

Tras esta idea, aparecieron las primeras implementaciones de sistemas criptográficos de clave pública. Las dos primeras fueron: Merkle-Hellman y Rivest-Shamir-Adleman (RSA). Con el tiempo, aparecieron nuevas ideas, pero en general, todas las implementaciones hasta 1988 podían agruparse en los siguientes dos grupos:

1. PKC's basados en problemas teórico-numéricos difíciles, como RSA, Goldwasser-Micali o Rabin.
2. PKC's basados en el problema de la mochila, como Merkle-Hellman o Shamir.

En lo que respecta a los primeros, no se conocían ataques eficientes. Sin embargo, ya por esa época el criptosistema de Merkle-Hellman había sido comprometido por el ataque de Shamir. Incluso, se empezaron a proponer nuevas implementaciones de ataques contra los PKC's basados en el problema de la mochila, los cuales comenzaron a resultar inseguros.

Por si no fuera suficiente, en 1985 se publicó uno de los algoritmos protagonistas del capítulo 5, el ataque de Lagarias-Odlyzko. Como hemos explicado, este ataque es distinto a todos los anteriores, ya que no realiza ninguna suposición sobre cómo se construye el criptosistema y, por tanto, puede aplicarse a cualquier sistema criptográfico de tipo mochila cuya densidad sea baja.

6.2. Preludio al criptosistema

Presentamos en esta sección, el nuevo criptosistema de tipo mochila [5], que posee una alta densidad y una base completamente distinta al resto de PKC's basados en el mismo problema. La construcción subyacente utiliza un resultado de R.C. Bose y S. Chowla [3] sobre representaciones únicas de sumas en secuencias finitas "densas".

El método de generación de claves requiere computar logaritmos discretos en cuerpos finitos. Una vez este paso sea realizado, el cifrado es muy rápido (tiempo lineal) y el descifrado también es razonablemente rápido (equivalente a RSA). Por tanto, la creación de claves es el paso más complicado, y aunque no se conocen algoritmos para calcular logaritmos discretos en tiempo polinómico, sí se conocen algoritmos prácticos para algunos casos particulares (por ejemplo, el algoritmo de Pohlig-Hellman [20]).

Es importante mencionar que todos los PKC's basados en problemas teórico-numéricos conocidos por entonces, eran como máximo, tan difíciles como la factorización, y por lo tanto, todos eran reducibles al problema de tomar logaritmos discretos en módulos compuestos. De este modo, en caso de que el problema del logaritmo discreto se volviese manejable (provocando que todos los PKC's basados en problemas teórico-numéricos fuesen inseguros), la creación de este sistema sería más sencilla, incluso para tamaños de mochilas mayores.

En su momento, los autores creyeron que un sistema de este tamaño sería capaz de frustrar tanto los ataques de baja densidad como los de búsqueda exhaustiva. Sin embargo, 10 años después de su publicación en 1988, Vaudenay publicó un artículo [25] en el que recuperaba mensajes cifrados por este método usando algoritmos operativos con tiempo de ejecución polinomial, para un rango de parámetros bastante grande (incluyendo los propuestos por Chor y Rivest). A pesar de esto, actualmente se considera seguro para otros rangos de parámetros.

6.3. Descripción del criptosistema

Procedemos a continuación a explicar paso a paso el método de Chor-Rivest basado en el problema de la mochila. Comenzamos con el algoritmo de generación de claves, para posteriormente explicar el cifrado y descifrado del método. Veremos que el tamaño de la clave en este criptosistema depende del tamaño de los parámetros considerados originalmente.

Algorithm 11 Algoritmo de generación de claves de Chor-Rivest

Input: Potencia de un primo $q = p^{\lambda}$, y entero positivo $h \leq q$, de manera que el cálculo de logaritmos en $\mathbb{F}_{q^h}$ pueda realizarse eficientemente (por ejemplo, tomando q de forma que el orden de $\mathbb{F}_{q^h}$ tenga solamente factores primos pequeños [20]).
Output: Clave pública pk y clave privada sk.

1: Elegir una raiz $t \in \mathbb{F}_{q^h}$ de un polinomio mónico irreducible de grado h, $F(x) \in \mathbb{F}_q[x]$. Cualquier elemento de $\mathbb{F}_{q^h}$ puede representarse como un polinomio en t con grado $\leq h$ y coeficientes en $\mathbb{F}_q$.
2: Seleccionar un generador g, del grupo multiplicativo $\mathbb{F}^*_{q^h}$. Cabe destacar, que el orden de $\mathbb{F}^*_{q^h}$ es $n = q^h - 1$.
3: Calcular los siguientes q logaritmos:

$$a_i = log_g(t + \alpha_i)$$

para todo $\alpha_i \in \mathbb{F}_q$, $0 \leq i \leq q - 1$.
4: Reordenar los elementos a_i por una permutación aleatoria:

$$\pi : \{0, 1, ..., q - 1\} \rightarrow \{0, 1, ..., q - 1\}$$

de manera que $b_i = a_{\pi(i)}$.
5: Generar ruido, considerando un número entero elegido al azar $0 \leq r \leq q^h - 2$, y luego:

$$c_i \equiv (b_i + r) \bmod (q^h - 1), \text{ con } i = 0, 1, ..., q - 1$$

6: Revelar la clave pública $pk = (c_0, ..., c_{q-1}, q, h)$ y esconder la clave privada $sk = (t, g, \pi, r)$.

Algorithm 12 Algoritmo de cifrado de Chor-Rivest

Input: Mensaje $m = (x_0, ..., x_{q-1})$ con exactamente h bits a 1 y clave pública $pk = (c_0, ..., c_{q-1}, q, h)$.
Output: Mensaje cifrado y.

1: Calcular el correspondiente mensaje cifrado aplicando:

$$y = \sum_{i=0}^{q-1} x_i \cdot c_i \pmod{q^h - 1}$$

2: Devolver el mensaje cifrado y.

Algorithm 13 Algoritmo de descifrado de Chor-Rivest

Input: Mensaje cifrado y (suponiendo que proviene de un mensaje de q bits con peso h) y clave privada $sk = (t, g, \pi, r)$.
Output: Mensaje descifrado $m = (x_0, ..., x_{q-1})$ con exactamente h bits a 1.

1: Calcular:
$$y' \equiv y - h \cdot r \pmod{q^h - 1}.$$

2: Obtener $g^{y'}$ escrito como polinomio en x. Existe un único polinomio $Q(x) \in \mathbb{F}_{q^h}[x]$ con grado $\leq h - 1$, tal que:
$$Q(x) \equiv g^{y'} \pmod{F(x)}$$
Sea $I = \{i_1 < \cdots < i_h\}$ el conjunto de índices con los bits correspondientes igual a 1, esto es, $x_{i_1} = \cdots = x_{i_h} = 1$, entonces:
$$g^{y'} = g^y \cdot g^{-hr} = g^{\sum_{i=0}^{q-1} x_i c_i} \cdot g^{-hr} = \prod_{i \in I} g^{c_i - r} = \prod_{i \in I} g^{b_i} = \prod_{i \in I} g^{a_{\pi(i)}} = \prod_{i \in I} (t + \alpha_{\pi(i)})$$

3: Como el polinomio de grado h, $F(x) + Q(x)$ factoriza linealmente en el cuerpo $\mathbb{F}_q$, entonces:
$$F(x) + Q(x) = \prod_{i \in I} (x + \alpha_{\pi(i)})$$

4: Sustituir los valores $\alpha_0, ..., \alpha_{q-1} \in \mathbb{F}_q$ para obtener las h raices de ese polinomio. Además, si las raices son $\alpha_{j1}, ..., \alpha_{jh}$, entonces aplicando la permutación inversa π^{-1} a esos índices de raíces, $\pi^{-1}(j_l) = i_l$, se obtienen los subíndices del mensaje con términos iguales a 1.

5: Devolver el mensaje obtenido $m = (x_0, ..., x_{q-1})$.

6.4. Análisis del criptosistema

Como ya hemos comentado, existen varios ataques conocidos para vulnerar el criptosistema de Chor-Rivest, pero en la práctica solo tienen éxito para los parámetros originalmente propuestos por los autores. A continuación, mencionamos algunos de ellos.

Chor y Rivest [5] describen tanto ataques especializados, donde el atacante conoce partes de la clave secreta del destinatario, como el diseñado por Goldreich y Odlyzko; como ataques generales, donde solo se conoce la clave pública. Entre estos últimos, se incluyen el ataque de fuerza bruta diseñado por Brickell [4], los ataques por baja densidad al problema de la mochila como el realizado por Lagarias y Odlyzko [15], y otros como el de Schnorr y Hörner [21].

Las parejas de parámetros (q, h) propuestas originalmente por Chor-Rivest fueron $(197, 24)$, $(211, 24)$, $(3^5, 24)$ y $(2^8, 25)$, cuyo número de dígitos son 56, 56, 58 y 60. Si el número de

cifras es suficientemente pequeño, el ataque de Schnorr-Hörner es capaz de romper el criptosistema.

Tal y como hemos indicado, Vaudenay fue el primero en romperlo realmente para un conjunto mucho más amplio de parámetros (incluidos los propuestos por los autores). Su ataque es remarcable, ya que utiliza el hecho de que el criptosistema de Chor-Rivest tiene claves secretas equivalentes, y es capaz de recuperar una de ellas. Una clave equivalente, por lo general, difiere de la clave secreta original pero funciona de la misma manera. Todos los usuarios pueden usar los mismos valores para q y h, ya que el riesgo de colisiones, es decir, que dos usuarios tengan la misma contraseña, es extremadamente bajo.

En realidad, como la clave privada está formada por el conjunto $\{t, g, \pi, r\}$, existen:

$$h \cdot \phi(q^h - 1) \cdot (q^h - 2) \cdot q!$$

diferentes claves privadas, de las cuales:

$$h \cdot q(q-1)$$

son equivalentes. Donde hemos denotado como ϕ a la función de Euler. Por tanto, el número de claves no equivalentes es:

$$n_{q,h} = \phi(q^h - 1) \cdot (q^h - 2) \cdot (q-2)!$$

Así, [9] explica que para k usuarios, el número de colisiones posibles es:

$$\frac{(n_{q,h}-1)(n_{q,h}-2)\cdots(n_{q,h}-k+1)}{n_{q,h}^k} \leq \frac{1}{n_{q,h}}$$

que sabemos por su cota, que es un valor pequeño. Por ejemplo, para la pareja de parámetros $q = 197$ y $h = 24$, obtenemos que su número máximo de colisiones es $\leq 0.16155 \cdot 10^{-472}$.

En resumen, todos los ataques descritos son capaces de romper el criptosistema de Chor-Rivest solamente para los parámetros originales y para parámetros que tienen ciertas propiedades (como el caso de Vaudenay). Esto abre la puerta a explorar nuevos parámetros situados en rangos más amplios, o seleccionados específicamente para eludir los ataques conocidos.

En el artículo [9], se menciona que se pueden hallar distintos pares de parámetros (q, h) que verifican las condiciones establecidas por Chor-Rivest y que refuerzan la resistencia del criptosistema ante los ataques conocidos. Verifican las siguientes condiciones:

1. $2 \leq h \leq q$, con q un número primo, y h un número primo o el cuadrado de un número primo.
2. El número de dígitos de n debe ser $t(q, h) \geq 36$.
3. La longitud en bits de la clave pública debe satisfacer $l(q, h) < 15.000$.

4. La densidad de la mochila debe ser $d(q,h) > 1$.

5. $u(n)$ debe tener como máximo 18 dígitos decimales.

Finalmente, remarcamos que la pareja $(1123, 13)$ forma la clave pública de mayor tamaño, del orden de 150 kilobits.

Conclusiones

Durante el desarrollo de este trabajo, hemos realizado un análisis de los fundamentos teóricos y prácticos de algunos criptosistemas de clave pública, poniendo especial atención al criptosistema de Merkle-Hellman. Dado que este criptosistema se basa en el problema de la mochila, hemos abarcado una gran variedad de conceptos necesarios para su compresión, que van desde nociones como las máquinas de Turing, hasta cuestiones tan fundamentales como el problema P vs NP.

A lo largo de este proyecto, además de explicar toda la teoría matemática apoyada de ejemplos esclarecedores, hemos destacado la relevancia práctica de los criptosistemas mediante su implementación. Esto nos ha permitido analizar los resultados obtenidos y compararlos con los teóricos esperados. De esta manera, hemos conseguido plasmar todo el conocimiento estudiado en aplicaciones concretas.

Asimismo, hemos abordado de manera firme los ataques principales a los criptosistemas estudiados, entendiendo que resulta casi imposible garantizar la total seguridad de un sistema criptográfico. Incluso en la actualidad, persistimos en la búsqueda de dicho criptosistema, especialmente ante el desafío que representan los algoritmos cuánticos, los cuales ponen en riesgo a muchos de los criptosistemas actuales.

Para finalizar, este trabajo de fin de grado ha supuesto el primer acercamiento a la actividad profesional, y como tal, este proyecto queda abierto ante futuras investigaciones y desarrollos en el campo de la criptografía y de la computación cuántica.

Bibliografía

Las referencias se listan por orden alfabético. Aquellas referencias con más de un autor están ordenadas de acuerdo con el primer autor.

[1] Automatas. Teoría de autómatas. http://teoriaautomatas.blogspot.com/2012/02/turing.html. RAI 2012 Universidad Carlos III. Recurso online. Accedido el 23/11/2023.

[2] Jingguo Bi, Xianmeng Meng, and Lidong Han. Cryptanalysis of two knapsack public-key cryptosystems. *IEEE Transactions on Information Theory*, IT-34(5), Septiembre 1988.

[3] R.C. Bose and S. Chowla. Theorems in the additive theory of numbers. *Commentarii Mathematici Helvetici*, 1962.

[4] Ernest F. Brickell. Solving low density knapsacks. *Springer*, 1984. Proceedings of Crypto '83 (Plenum Press, New York, 1984). pages 243-256.

[5] Benny Chor and Ronald L. Rivest. A knapsack-type public key cryptosystem based on arithmetic in finite fields. *IEEE Transactions on Information Theory*, 34(5), Septiembre 1988.

[6] Patricia de Armas González. El algoritmo lll. *La Laguna*, Julio 2016.

[7] Dean and Walter. Computational Complexity Theory. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2021 edition, 2021.

[8] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6), Noviembre 1976.

[9] Raul Durán Díaz, Luis Hernández-Álvarez, Luis Hernández Encinas, and Araceli Queiruga-Dios. Chor-rivest knapsack cryptosystem in a post-quantum world. *Springer Nature Switzerland*, Agosto 2021.

[10] Matthijs J. Coster et al. Improved low-density subset sum algorithms. *Comput Complexity 2*, 1992.

[11] Helen Fouché Gaines. *Cryptanalysis a Study of Ciphers and their Solution*. Dover Publications, INC. New York, 1956. pages 68-69.

[12] Clay Maths Institute. The millennium prize problems. https://www.claymath.org/millennium-problems/. Recurso online. Accedido el 23/11/2023.

[13] A. K. Lenstra, H. W. Lenstra Jr, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, Julio 1982.

[14] RSA Laboratories. *RSA Laboratories' Frequently Asked Questions About Today's Cryptography, Version 4.1*. RSA Security Inc., Mayo 2000. pages 34-35.

[15] J. C. Lagarias and A. M. Odlyzko. Solving low-density subset sum problems. *Journal of the Association for Computing Machinery*, 32(1), Enero 1985.

[16] Rudolf Lidl and Harald Niederreiter. Finite fields. *Encyclopedia of Mathematics and its applications*, 1983.

[17] F. J. MacWilliams and N. J. A. Sloane. The theory of error-correcting codes. https://doi.org/10.1137/1022103. Recurso online. Accedido el 23/11/2023.

[18] Ralph C. Merkle and Martin E. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory*, IT-24(5), Septiembre 1978.

[19] A. M. Odlyzko. The rise and fall of knapsack cryptosystems. *AT&T Bell Laboratories Murray Hill, New Jersey o7974*, pages 3–4, 1998.

[20] Stephen C. Pohlig and Martin E. Hellman. An improved algorithm for computing logarithms over gf(p) and its cryptographic significance. *IEEE Transactions on Information Theory*, 24(1):106–110, Enero 1978.

[21] C. P. Schnorr and H. H. Hörner. Attacking the chor-rivest cryptosystem by improved lattice reduction. *Springer*, 1995. International Conference on the Theory and Applications of Cryptographic Techniques (Springer, New York, 1995), pages 1–12.

[22] Adi Shamir. A polynomial-time algorithm for breaking the basic merkle-hellman cryptosystem. *IEEE Transactions on Information Theory*, IT-30(5), Septiembre 1984.

[23] Mark Stamp. Lattice reduction attack on the knapsack. pages 1-2.

[24] Harshvardhan Tiwari and Krishna Asawa. Cryptographic hash function: An elevated view. *European Journal of Scientific Research*, 43(4):452–453, 2010.

[25] Serge Vaudenay. Cryptanalysis of the chor-rivest cryptosystem. *Springer*, 1998. Advances in Cryptology CRYPTO'98. pages 243-256.

[26] Joachim von zur Gathen. *CryptoSchool*. Springer, 2016. pages 13-56.

[27] Weisstein and Eric W. Nondeterministic turing machine. https://mathworld.wolfram.com/NondeterministicTuringMachine.html. Recurso online. Accedido el 23/11/2023.

[28] Weisstein and Eric W. Np-problem. https://mathworld.wolfram.com/NP-Problem.html. Recurso online. Accedido el 23/11/2023.

[29] Weisstein and Eric W. P-problem. https://mathworld.wolfram.com/P-Problem.html. Recurso online. Accedido el 23/11/2023.

[30] Weisstein and Eric W. Turing machine. https://mathworld.wolfram.com/TuringMachine.html. Recurso online. Accedido el 23/11/2023.