

UNIVERSIDAD ABIERTA INTERAMERICANA
Facultad de Tecnología Informática



Carrera: Licenciatura en Matemática

CRIPTOGRAFÍA

Un modelo compacto de criptografía asimétrica empleando anillos no conmutativos.

Autor: María Fernanda López
Directora: Dra. Samira Abdel Masih

***TESIS PRESENTADA PARA OPTAR AL TÍTULO DE
LICENCIADO EN MATEMÁTICA***

- Julio 2020 -

Firmas del jurado

Agradecimientos

Me gustaría empezar esta tesis agradeciendo a todas aquellas personas que me ayudaron y acompañaron durante esta hermosa etapa.

Agradezco especialmente a Samira, mi directora, por aceptar mi propuesta de ser su tesista, por su compromiso, la buena predisposición y la gran ayuda que me brindó desde el primer momento que comenzamos a trabajar en esta Tesis. Además de una gran docente fue una excelente guía en esta última parte del camino.

A mi familia que me acompañó y apoyó en todo momento durante toda la carrera, y que, gracias a su infinita paciencia y consejos hoy logro culminar la carrera.

A todos mis amigos de la vida, del colegio, del trabajo y a mis compañeros de la facultad por su compañía en el transcurso de la carrera.

A todo el jurado por tomarse el tiempo de leer esta tesis.

A la UAI y a todos los docentes que tuve a lo largo de la carrera, quienes aportaron mucho a mi formación como matemática e hicieron de mi paso por la Facultad una experiencia hermosa.

Muchas gracias a todos por creer en mí...

Resumen

En este trabajo analizaremos la publicación presentada por Pedro Hecht [2], en la que desarrolla un modelo de Criptografía Asimétrica: el Protocolo de Intercambio de Claves Diffie-Hellman utilizando estructuras algebraicas no conmutativas. En este caso, el autor utiliza el anillo no conmutativo de matrices.

Su objetivo es hallar un modelo que sea criptográficamente seguro pero capaz de emplear recursos computacionales escasos. Este tipo de modelos se los define como compactos, y utilizan procesadores de sólo 16 bits.

De este modo, el algoritmo permitirá una eficaz operación en plataformas de baja potencia, como por ejemplo, tarjetas inteligentes o teléfonos celulares.

A los efectos de facilitar al lector la comprensión de los temas, los ordenamos y expusimos de la siguiente manera:

- Capítulo 1: Introducimos las nociones básicas de Criptografía y haremos mención de sus antecedentes históricos.
- Capítulo 2: Presentamos los conceptos matemáticos previos que serán necesarios para entender el desarrollo de este trabajo.
- Capítulo 3: Describimos algunos de los primeros métodos criptográficos.
- Capítulo 4: Explicamos los detalles del algoritmo criptográfico Diffie-Hellman.
- Capítulo 5: Desarrollamos el algoritmo criptográfico Diffie-Hellman empleando el anillo no conmutativo de matrices.
- Capítulo 6: Elaboramos las Conclusiones.

Palabras claves:

Criptografía, Criptografía Asimétrica, Aritmética Modular, Diffie-Hellman, Anillos no Conmutativos.

Índice

Introducción	7
Introducción	7
Capítulo 1: Introducción a la Criptografía	8
¿Qué es la Criptografía?	9
Antecedentes históricos de la Criptografía	10
Usos de la Criptografía	13
Clasificación de la Criptografía	13
Capítulo 2: Conceptos matemáticos previos	16
Aritmética modular	17
Divisibilidad	17
Congruencia	18
Conjunto $\mathbb{Z}_n$	19
Operaciones en $\mathbb{Z}_n$	19
Cálculo de potencias modulares	23
Logaritmos discretos	27
El problema de los logaritmos discretos	29
Almacenamiento de datos en un computador	29
Orden de complejidad computacional de un algoritmo	31
Estructuras algebraicas	36
Capítulo 3: Primeros métodos criptográficos	38
La Escítala	39
Codificación del alfabeto	42

<i>Vulnerabilidad del Método de Julio César Mejorado</i>	42
<i>Descripción del Método de Matrices y Digrafas</i>	
<i>Ventaja del Método de Matrices y Digrafas con respecto a los Métodos de Julio César y Julio César Mejorado</i>	44
	49
Capítulo 4: Algoritmo de Diffie-Hellman	51
<i>Algoritmo de Diffie-Hellman</i>	52
<i>Procedimiento del Algoritmo</i>	53
<i>Seguridad del Algoritmo de Diffie-Hellman</i>	54
<i>Una desventaja del Algoritmo de Diffie-Hellman</i>	54
Capítulo 5: Algoritmo de Diffie-Hellman empleando anillos no conmutativos	57
<i>El Algoritmo de Diffie-Hellman aplicado a anillos no conmutativos</i>	58
<i>Desventajas de los algoritmos criptográficos basados en estructuras algebraicas no conmutativas</i>	58
<i>Modelo criptográfico compacto</i>	59
<i>El anillo de matrices a coeficientes en $\mathbb{Z}_{256}$</i>	59
<i>Producto de un entero por un elemento de un Anillo</i>	61
<i>Potencia de un elemento de un Anillo</i>	62
<i>Polinomios definidos en un Anillo</i>	64
<i>El conjunto $\mathbb{Z}_{\geq 0}[A]$</i>	64
<i>Procedimiento del algoritmo Diffie-Hellman usando anillos no conmutativos</i>	65
<i>Protocolo de Diffie-Hellman con matrices</i>	68
<i>Análisis del tiempo y espacio de memoria requerido para la ejecución del Algoritmo de Diffie-Hellman aplicando matrices</i>	72
<i>Consideraciones acerca de la seguridad computacional del protocolo propuesto</i>	82
Capítulo 6: Conclusiones	87
Referencias	89

Introducción

Secretos. Todos los tenemos. La vida de cualquier persona involucra en algún momento (me atrevería a decir *a diario*) algún secreto, algo que quiera o tenga que ocultar. Y aunque usted esté tentado a decir "no, yo no tengo nada que ocultar", es muy probable que utilice un *password* para acceder a una computadora, a una correspondencia electrónica, a una tarjeta de crédito o a una máquina de la que retira dinero en efectivo. Hasta la combinación de una caja de seguridad o de un candado común y corriente requiere de un *password*. Y ni qué hablar de hacer alguna transacción por Internet que involucre alguna identificación personal.

¿Cuántas veces le contaron o leyó que hay gente interesada en interceptar datos que usted envía por Internet para usarlos maliciosamente? ¿Cuántas veces dudó en enviar, también por Internet, el número de su documento, o su dirección postal, o el número de su tarjeta de crédito o de su cuenta bancaria, por temor a que fueran robados? ¿Cuántas veces pensó que hay gente que puede leer el contenido de sus mensajes de correo electrónico sin que usted lo advierta?

¿Cuántas veces oyó que, para poder garantizar ciertas condiciones mínimas de seguridad, hace falta usar Matemática? Ahora bien, ¿qué "Matemática" hay que usar? ¿Y cómo? ¿Cuán seguro es?

La Matemática que utilizaremos para crear *passwords* se basará en el uso de potencias modulares, tema que veremos a lo largo de este trabajo.

Apoyándose en este concepto matemático, en 1976 dos científicos norteamericanos, Whitfield Diffie (1944) y Martin Hellman (1945), de la Universidad de Stanford, introdujeron una idea revolucionaria: utilizar dos claves para generar un *password*.

Una de las claves es "pública" y la otra "privada".

¿Qué quiere decir esto? Que la "clave pública" puede ser conocida por todos, mientras que la "clave privada" es conocida por una única persona. Ambas claves se combinan para dar origen al *password*, el cual es conocido sólo por dos personas.

La seguridad de este método radica, como lo comprobaremos, en la dificultad para calcular ciertos logaritmos discretos.

En los últimos años existe una fuerte tendencia al uso de logaritmos discretos aplicados a estructuras algebraicas no conmutativas.

El presente trabajo se basa en el uso del Algoritmo de Diffie-Hellman aplicado al *anillo no conmutativo de matrices* $M_4[\mathbb{Z}_{256}]$ es decir, matrices de orden 4 a coeficientes en $\mathbb{Z}_{256}$. Este tipo de estructuras no requieren el empleo de bibliotecas numéricas de precisión extendida, ya que cada matriz ocupa una memoria de 16 bytes y todas las operaciones numéricas se efectúan dentro del anillo $\mathbb{Z}_{256}$. Esto otorga un gran beneficio: el algoritmo puede ser ejecutado en procesadores de bajo poder computacional y memoria RAM reducida.

Capítulo 1

Introducción a la Criptografía

Capítulo 1: Introducción a la Criptografía

En este capítulo definiremos qué es la Criptografía, sus clasificaciones y sus antecedentes históricos.

La palabra Criptografía proviene del griego “*kryptos*” que significa oculto, y “*graphia*”, que significa escritura. Entre las disciplinas que engloba cabe destacar la Informática, la Teoría de Números y la Matemática Discreta.

Puede ser usada para prevenir el acceso y uso no autorizado de los recursos de una red o sistema informático y para prevenir a los usuarios la denegación de los servicios a los que sí están permitidos.

Como es conocido, los aliados en la Segunda Guerra Mundial lograron decodificar los mensajes de los alemanes y de esa forma lograron invadir Normandía en 1944 y, a partir de ahí, consiguieron el triunfo final. Es decir, la encriptación de mensajes es un tema de alta sensibilidad.

Modernamente, la Criptografía es la metodología para proveer la seguridad de las redes telemáticas, incluyendo la identificación de entidades y autenticación, el control de acceso a los recursos, la confidencialidad de los mensajes transmitidos, la integridad de los mensajes y su no repudio. Su definición formal es la siguiente:

Definición 1.1: Criptografía

La Criptografía es la ciencia que se ocupa de crear algoritmos para ocultar mensajes y de este modo, evitar que sean leídos por ciertas personas.

La criptografía se compone de dos pasos: **Encriptado** y **Desencriptado**.

Encriptado: Consiste en convertir un mensaje (texto llano) a un texto cifrado.

Desencriptado: Consiste en recuperar el mensaje original a partir del texto cifrado.

El encriptado lo realiza el emisor del mensaje. Mientras que el desencriptado lo lleva a cabo el receptor del mensaje.

La Criptografía, mediante el ocultamiento de la información proporciona:

- **Confidencialidad:** Garantiza que sólo las personas autorizadas tengan acceso a la información.
- **Autenticación:** Verifica que la información provenga del lugar indicado y que ésta no haya sido modificada.
- **Integridad:** Detecta que la información no haya sido alterada por alguien no autorizado.
- **Control de Acceso:** Restringe el acceso a la información.
- **No repudio:** Garantiza la autoría del mensaje enviado.

Antecedentes históricos de la Criptografía

Entre el Antiguo Egipto e Internet, los criptogramas (los mensajes cifrados) han protagonizado buena parte de los grandes episodios históricos. Abundan en los textos diplomáticos de toda época, indispensables para las órdenes militares y los ejércitos modernos en tiempos de guerra y, por supuesto, esenciales en la actividad de los espías. Hoy en día, con las nuevas tecnologías, el uso de la criptografía se ha extendido más allá de su tradicional esfera estatal o política, y es vital también para la actividad diaria de las empresas y ciudadanos particulares.

El primer sistema criptográfico de la historia fue la **Escítala**. Era utilizada por los espartanos en el año 400 a.C. para el envío de mensajes escritos. Está formada por dos varas de igual grosor y de una tira de cuero o papiro sobre la cual se escribe el mensaje. Detallaremos su forma de emplearla en el Capítulo 3.



Figura 1.1: Escítala

Fuente: <https://es.wikipedia.org/wiki/Esc%C3%ADtala>

El escritor griego Polybios (200-118 A.C.) describe un curioso sistema de transmisión de señales, llamado **Polibio**, que se puede considerar como la primera “cifra”, es decir, el primer modo de escribir con caracteres arbitrarios de tal manera que sólo lo pueda entender el que tenga la clave. Consta de una tabla (de 5×5) donde se escribe el correspondiente alfabeto.

	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I,J	K
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

Figura 1.2: Tabla de Polibio

El Método de Julio César (100 a.C.-44 a.C) es una de las técnicas de cifrado más simples y más usadas. Es un tipo de cifrado por sustitución monoalfabética en el que una letra en el texto original es reemplazada por otra letra que se encuentra un número fijo de posiciones más adelante en el alfabeto. Su desarrollo se verá en el Capítulo 3.

En el siglo XV Leon Battista Alberti (1404-1472) inventó un sistema de sustitución polialfabética, conocido como **Cifrado Vigenere**, por haber sido atribuido por error a Blaise de Vigeniere (1523-1596). Con este sistema cada letra tiene una correspondencia única, haciendo más difícil el descifrado.

Hay un período donde la criptografía cobró vital importancia: en la Segunda Guerra Mundial (1939-1945), en la que el cifrado y el descifrado de códigos se convirtió en otro frente más de lucha.

Hubo un sistema de cifrado utilizado por Alemania y que tuvo en jaque a los aliados, sobre todo, en el Atlántico Norte, donde los convoys de material procedente de Estados Unidos caían presa de los submarinos alemanes que se comunicaban entre sí utilizando el código que generaba uno de los inventos más fascinantes de esa época: **la máquina Enigma**.



Figura 1.3: Máquina Enigma

Fuente: [https://es.wikipedia.org/wiki/Enigma_\(máquina\)](https://es.wikipedia.org/wiki/Enigma_(máquina))

¿En qué consistía la máquina Enigma?

La máquina Enigma era un dispositivo electromecánico, es decir, tenía una parte eléctrica y otra mecánica. El mecanismo consistía en una serie de teclas, con las letras del alfabeto, al igual que una máquina de escribir, que en realidad eran interruptores que accionaban los dispositivos eléctricos y hacían mover unos cilindros rotatorios. El funcionamiento era bastante sencillo. El operador tenía que teclear las letras de su mensaje y anotar las letras que devolvía la máquina (a través de un alfabeto que se iba iluminando). El código a usar se fijaba con las posiciones de los cilindros que constaban, cada uno, de 26 cables que se conectaban al teclado pero, con la particularidad de que el primer cilindro giraba un veintiseisavo de vuelta después de cada pulsación, de tal manera que la posición de las conexiones iba cambiando con cada entrada del teclado, obteniendo un cifrado polialfabético. Además, para dar mayor robustez, el segundo cilindro sólo daba un giro cuando el primero había completado 26 giros y el tercero cuando el segundo había dado sus correspondientes 26 y añadió la

posibilidad de que los rodillos pudiesen ser intercambiados de posición, de manera que el número de posibilidades aumentase hasta tener 105.456 alfabetos.

Además, el sistema contaba con 6 cables de conexión que también permitían introducir modificaciones, dado que podrían conectarse a 26 lugares (representando a las 16 letras del alfabeto de Enigma). Esto producía 100.391.791.500 maneras distintas de conectar los cables que, unidos a los 105.456 alfabetos arrojaba

3.283.883.513.796.974.198.700.882.069.882.752.878.379.955.261.095.623.

685.444.055.315.226.006.433.616.627.409.666.933.182.371.154.802.769.920.000.000.

000

posibilidades distintas de codificación.

¿Y cómo se logró vencer a Enigma?

Fue vencida gracias a varios factores:

- Estas databan de 1923 y eran modelos comerciales que se distribuyeron por todo el mundo. Si bien es cierto que las máquinas se fueron complicando en sus versiones militares, el principio de funcionamiento ya era conocido.
- La codificación de un mensaje en Enigma obligaba a los operadores a introducir 3 letras, dos veces, al iniciar el mensaje, una especie de bandera. La Luftwaffe (fuerza aérea de Alemania) no modificaba esta secuencia y, por tanto, era un patrón que siempre se repetía y que fue aprovechado por Marian Rejewski (1905-1980) que fue capaz de descifrar el código gracias a unas máquinas electromecánicas denominadas bombas criptológicas que eran unas máquinas Enigma de procesamiento en paralelo y que buscaban las codificaciones posibles. Los criptógrafos polacos trabajaron junto a los británicos en Bletchley Park (instalación militar ubicada en Inglaterra) para descifrar los códigos alemanes (en este equipo se encontraba Alan Turing (1912-1954)).
- El 9 de mayo de 1941 la Royal Navy capturó el submarino alemán U-110 (a cuya tripulación le fallaron las cargas explosivas para hundirlo). Así, lograron apoderarse de una máquina Enigma hallada en el submarino, junto con el preciado libro de claves. Esta captura se mantuvo en secreto y se hizo creer a la opinión pública que el submarino había sido hundido, para que las claves no fuesen cambiadas.

La suma de estos factores obtuvo como resultado el descifrado de los mensajes de Enigma y, por tanto, la drástica disminución de las pérdidas Aliadas en el Atlántico Norte. Ante las sucesivas derrotas, los Alemanes evolucionaron Enigma y crearon una nueva máquina, la M4 pero fue vencida gracias a Colossus, un computador diseñado para descifrar los códigos alemanes.

Usos de la Criptografía

La Criptografía cuenta con 3 usos: **Cifrar**, **Autenticar** y **Firmar**.

Cifrar: Como ya hemos dicho, siempre hay cierta información que no queremos que sea conocida más que por las personas que nosotros queramos. En esto nos ayuda el cifrado. Cifrando un mensaje hacemos que este no pueda ser leído por terceras personas consiguiendo así la tan deseada privacidad.

Autenticación: Otra de las necesidades que surgen con la aparición de internet es la necesidad de demostrar que somos nosotros y que el emisor es quien dice ser. Un método de autenticación puede ser el propio cifrado. Si ciframos un mensaje con una clave sólo conocida por nosotros, demostrando que somos quien decimos ser, el receptor podrá constatar nuestra identidad descifrándolo. Esto se puede conseguir mediante clave simétrica (el receptor tiene que estar en posesión de la clave empleada) o usando clave asimétrica en su modo de autenticación.

Firmar: Dados los trámites que podemos realizar hoy en día a través de internet, se hace necesaria la aparición de la firma digital. Igual que firmamos un documento, la firma digital nos ofrece la posibilidad de asociar una identidad a un mensaje.

Para la firma digital se utiliza clave asimétrica (dos claves: una privada y otra pública). Lo que se cifra con la clave privada (que sólo nosotros conocemos) se puede descifrar con la pública. De esta forma al cifrar con nuestra clave privada demostramos que somos nosotros.

Clasificación de la Criptografía

En su clasificación dentro de las ciencias, la Criptografía proviene de una rama de las Matemáticas, que fue iniciada por el matemático Claude Elwood Shannon (1916-2001) en 1948, denominada: “Teoría de la Información”. Esta rama de las ciencias se divide en: “Teoría de Códigos” y en “Criptología”. Y a su vez la Criptología se divide en Criptoanálisis y Criptografía, como se muestra en el siguiente esquema:

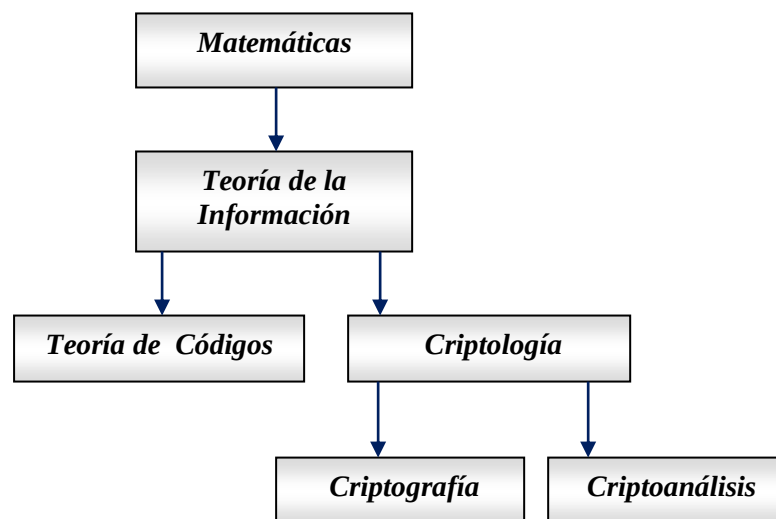


Figura 1.4: Origen de la Criptografía

En un sentido más amplio, la Criptografía es la ciencia encargada de diseñar funciones o dispositivos capaces de transformar mensajes legibles en mensajes cifrados.

En contraparte, el Criptoanálisis es la ciencia que estudia los métodos que se utilizan para, a partir de uno o varios mensajes cifrados, recuperar los mensajes originales.

En su clasificación histórica, se puede clasificar en dos: la **Criptografía Clásica** y la **Criptografía Moderna**:

La Criptografía Clásica es la que se utilizó desde la antigüedad hasta la mitad del siglo XX. También puede entenderse como la Criptografía no computarizada o no digitalizada. Los métodos utilizados eran variados, algunos muy simples y otros muy complicados de criptoanalizar para su época.

La Criptografía Moderna se inició después de tres hechos importantes: el primero, en 1940, con la publicación por parte de Shannon de la “*Teoría de la Información*”; el segundo, con la aparición en 1974 del sistema de cifrado DES (Data Encryption Standard) y finalmente, en 1976, con el estudio realizado por Whitfield Diffie y Martin Hellman referente a la aplicación de funciones matemáticas a un modelo de cifrado, denominado “*Cifrado de Llave Pública*”.

En su clasificación de acuerdo a las técnicas o métodos que se utilizan para cifrar los mensajes, se puede clasificar en dos: **Criptografía Simétrica** y **Criptografía Asimétrica**:

Criptografía Simétrica: es un método criptográfico monoclave, es decir, usa la misma clave para cifrar que para descifrar. Esto supone un grave problema a la hora de realizar el intercambio entre el emisor y el receptor, dado que si una tercera persona estuviese escuchando el canal, podría capturar la clave, siendo inútil el cifrado.

Es importante que la clave sea difícil de descubrir. En la actualidad, con la capacidad computacional disponible, se puede obtener la clave en cuestión de minutos u horas. Por esta razón este tipo de Criptografía ya no es segura.



Figura 1.5: Criptografía Simétrica

Ejemplos de Criptografía Simétrica son: Enigma, El Método de Julio César, el Método lineal ó de Julio César Mejorado y el Método de las Digrafías.

Criptografía Asimétrica: cuando se utilizan distintas claves para encriptar y para desencriptar. En este caso, la clave que se utiliza para encriptar puede ser pública (conocida por todos) y la clave para desencriptar debe ser privada, es decir, conocida sólo por el destinatario (ni siquiera es conocida por el remitente). Ofrece un abanico superior de posibilidades, pudiendo emplearse para establecer comunicaciones seguras por canales inseguros, puesto que únicamente viaja por el canal la clave pública, que sólo sirve para cifrar, o para llevar a cabo autenticaciones. Sin la clave privada (que no

es deducible a partir de la clave pública) un observador no autorizado del canal de comunicación será incapaz de descifrar el mensaje cifrado.



Figura 1.6: Criptografía Asimétrica

Algunos ejemplos de criptografía asimétrica son: Algoritmo de Diffie-Hellman, Algoritmo RSA (basa su seguridad en la dificultad de factorizar un número natural), Algoritmo DSA, Criptografía de Curva Elíptica (utiliza como función trampa la exponenciación modular, cuya función inversa es el logaritmo discreto).

A finales del siglo XX, el desarrollo de la Informática e Internet dieron lugar a grandes cambios en Criptografía. Por ejemplo, la gran cantidad de información a disposición de muchos usuarios hace necesario que los datos estén protegidos durante su almacenamiento, no sólo durante la transmisión. Es por eso que, actualmente, la Criptografía ha ampliado sus objetivos, además de la transmisión secreta de la información.

Capítulo 2

Conceptos matemáticos previos

Capítulo 2: Conceptos matemáticos previos

En este capítulo expondremos las nociones matemáticas que serán aplicadas en el desarrollo de este trabajo.

Definición 2.1: Aritmética modular

La Aritmética Modular es la parte de la aritmética que opera con números enteros, considerando el resto de la división de dichos números enteros por un número natural “n” fijo, llamado “el módulo de n”.

La aritmética modular fue creada por Gauss (1777-1855) para resolver de una manera más sencilla, las ecuaciones diofánticas. Estas ecuaciones fueron planteadas por Diofanto de Alejandría (200-284), las cuales son de la forma:

$$a x + b y = c$$

Donde $a, b, y c \in \mathbb{Z}$

incógnitas: $x, y \in \mathbb{Z}$

Ejemplo 2.2

La ecuación diofántica $3x + 6y = 18$ tiene infinitas soluciones enteras. Algunas de estas soluciones son:

$$x = 4, \quad y = 1$$

$$x = -6, \quad y = 6$$

$$x = 4, \quad y = -2$$

Los estudios de Gauss sobre este tipo de aritmética revolucionaron las matemáticas de principios de siglo XIX, ayudando a los matemáticos a descubrir estructuras que habían permanecido ocultas. Hoy en día la aritmética de congruencias, también llamada modular, es fundamental para la seguridad de Internet, donde se utilizan congruencias con cantidades que superan la de átomos del universo.

Definición 2.3: Divisibilidad

Sean $a, y b \in \mathbb{Z}$ con $b \neq 0$. Se dice que “b” divide a “a” y se escribe $b|a$ si $a = k \cdot b$ para algún $k \in \mathbb{Z}$. O equivalentemente, si al dividir a por b, el resto da 0.

Definición 2.4: Congruencia

Dado un número natural $n > 1$ y dos números a y $b \in \mathbb{Z}$. Se dice que a es congruente con b módulo n y se indica

$$a \equiv b \pmod{n}$$

si $n \mid (b - a)$ ó $n \mid (a - b)$

Ejemplo 2.5

- 1) $8 \equiv 4 \pmod{2}$ pues $2 \mid (8 - 4)$
- 2) $20 \not\equiv 3 \pmod{7}$ pues $7 \nmid (20 - 3)$
- 3) $56 \not\equiv 3 \pmod{5}$ pues $5 \nmid (56 - 3)$

La siguiente proposición enumera ciertas propiedades de la congruencia que aplicaremos en esta tesis:

Proposición 2.6: Sean $a, b, c \in \mathbb{Z}$

- 1) *Propiedad Reflexiva:*
 $a \equiv a \pmod{n}$
- 2) *Propiedad Simétrica:*
 $\forall a, b \in \mathbb{Z}, \text{ si } a \equiv b \pmod{n} \Rightarrow b \equiv a \pmod{n}$
- 3) *Propiedad Transitiva:*
 $\forall a, b, c \in \mathbb{Z}, \text{ si } a \equiv b \pmod{n} \wedge b \equiv c \pmod{n} \Rightarrow a \equiv c \pmod{n}$
- 4) $\forall a \in \mathbb{Z}, a \equiv 0 \pmod{n} \Leftrightarrow n \mid a$
- 5) Si $a \equiv b \pmod{n} \wedge c \equiv d \pmod{n} \Rightarrow a \cdot c \equiv b \cdot d \pmod{n}$
- 6) Si $a \equiv b \pmod{n} \wedge c \equiv d \pmod{n} \Rightarrow a + c \equiv b + d \pmod{n}$
- 7) Si $a \equiv b \pmod{n} \wedge c \neq 0 \Rightarrow a \cdot c \equiv b \cdot c \pmod{n}$
- 8) Si $a \equiv b \pmod{n} \wedge k \in \mathbb{N} \Rightarrow a^k \equiv b^k \pmod{n}$
- 9) $a \equiv b \pmod{n} \Leftrightarrow a \wedge b$ tienen el mismo resto en la división entera por n
- 10) $a \equiv r_a \pmod{n}$ donde r_a es el resto de la división entera de a por n
- 11) $a \equiv n + a \pmod{n}$

Demostración

Está detallada en [3]. ■

El conjunto $\mathbb{Z}_n$

De la Proposición 2.6 inciso 10), se sabe que si $a \in \mathbb{Z}$ y n es un natural mayor que 1, entonces $a \equiv r_a \pmod{n}$, donde r_a es el resto de la división entera de a por n . Esto permite clasificar a los números enteros según el resto de su división entera por n . Como los posibles restos son $0, 1, \dots, n-1$, surge la siguiente:

Definición 2.7: El conjunto $\mathbb{Z}_n$

Si n es un número natural mayor que 1, el conjunto $\mathbb{Z}_n$ es

$$\mathbb{Z}_n = \{ 0, 1, 2, \dots, n-1 \}.$$

A dicho conjunto se le definirán ciertas operaciones, a fin de dotarlo de una estructura algebraica conveniente.

Notación: utilizaremos la siguiente equivalencia: $a \equiv b \pmod{n} \Leftrightarrow a = b \text{ en } \mathbb{Z}_n$

Operaciones en $\mathbb{Z}_n$

Definición 2.8: Suma, Producto y Potencia en $\mathbb{Z}_n$

Sean $a, b, c \in \mathbb{Z}$. Entonces

1) Suma en $\mathbb{Z}_n$:

$$a + b = c \text{ (en } \mathbb{Z}_n) \Leftrightarrow c \text{ es el resto de la división entera de } (a + b) \text{ por } n.$$

2) Producto en $\mathbb{Z}_n$:

$$a \cdot b = c \text{ (en } \mathbb{Z}_n) \Leftrightarrow c \text{ es el resto de la división entera de } (a \cdot b) \text{ por } n.$$

3) Potencia en $\mathbb{Z}_n$: Si $n \in \mathbb{N}$ entonces:

$$a^n = \underbrace{a \cdot a \cdot \dots \cdot a}_{n \text{ veces}}$$

$$a^0 = 1$$

De la Definición anterior se deduce que

$$1) a + b = c \text{ (en } \mathbb{Z}_n) \Leftrightarrow a + b \equiv c \pmod{n}$$

$$2) a \cdot b = c \text{ (en } \mathbb{Z}_n) \Leftrightarrow a \cdot b \equiv c \pmod{n}$$

Definición 2.9: Elementos neutros para la suma y multiplicación en $\mathbb{Z}_n$

El neutro para la suma es el “0” pues $a + 0 = a \pmod{n} \forall a \in \mathbb{Z}_n$

El neutro para el producto es el “1” pues $a \cdot 1 = a \pmod{n} \forall a \in \mathbb{Z}_n$

Definición 2.10: El opuesto de $\mathbb{Z}_n$

Dado $a \in \mathbb{Z}_n$, el opuesto de a es un número $b \in \mathbb{Z}_n$ tal que $a + b = 0 \pmod{n}$

La siguiente proposición muestra como calcular el opuesto de un elemento de $\mathbb{Z}_n$

Proposición 2.11

Si $a \in \mathbb{Z}_n$ entonces su opuesto es $(n - a)$

Demostración

Como

$$a + (n - a) = n \equiv 0 \pmod{n}$$

Al aplicar la Propiedad Transitiva (Proposición 2.6, inciso 3) se obtiene:

$$a + (n - a) \equiv 0 \pmod{n}$$

Por lo tanto, por definición de la operación suma en $\mathbb{Z}_n$, se tiene que

$$a + (n - a) = 0 \pmod{n}$$

■

Notación: al opuesto de a lo simbolizaremos por “ $-a$ ”

Ejemplo 2.12

Sea $\mathbb{Z}_9 = \{ 0, 1, 2, 3, 4, 5, 6, 7, 8 \}$

En este caso, $n = 9$. Por lo tanto

Opuesto de 3:

$$(n - a) = 9 - 3 = 6$$

Opuesto de 7:

$$(n - a) = 9 - 7 = 2$$

Definición 2.13: Inverso en $\mathbb{Z}_n$

Dado $a \in \mathbb{Z}_n$ con $a \neq 0$, el inverso de a es un número $b \in \mathbb{Z}_n$ tal que $a \cdot b = 1$ en $\mathbb{Z}_n$

Notación: al inverso de a lo simbolizaremos por a^{-1} .

No todo elemento de $\mathbb{Z}_n$ tiene inverso, como se verá en el siguiente Ejemplo:

Ejemplo 2.14

Sea $\mathbb{Z}_6 = \{ 0, 1, 2, 3, 4, 5 \}$ indicar los elementos de $\mathbb{Z}_6$ que son invertibles.

Solución

Elaboraremos la siguiente tabla, que contendrá los productos entre dos elementos no nulos de $\mathbb{Z}_6$

•	1	2	3	4	5
1	1	2	3	4	5
2	2	4	0	2	4
3	3	0	3	0	3
4	4	2	0	4	2
5	5	4	3	2	1

Observando los datos de la tabla, se puede determinar qué elementos tienen inverso: Si una fila de la tabla contiene el 1, entonces el elemento que encabeza dicha fila tiene inverso. Además, su inverso es el elemento que encabeza la columna que contiene al 1. De este modo,

Elemento de $\mathbb{Z}_6$:	Inverso en $\mathbb{Z}_6$:
1	1
2	No tiene
3	No tiene
4	No tiene
5	5

El siguiente teorema nos indica en qué caso un elemento en $\mathbb{Z}_n$ es invertible.

Teorema 2.15: Teorema de Invertibilidad

Un elemento $a \in \mathbb{Z}_n$ es invertible $\Leftrightarrow \text{mcd}(a, n) = 1$

Demostración

Está detallada en [4]. ■

Corolario 2.16: Corolario del Teorema de invertibilidad en $\mathbb{Z}_n$

Si n es primo entonces todo elemento de $\mathbb{Z}_n$ es invertible

Demostración

Sea $a \in \mathbb{Z}_n$ con $a \neq 0$. Como n es primo, $\text{mcd}(a, n) = 1$

Por lo tanto, por el Teorema 2.15, resulta que a es invertible. ■

La siguiente Proposición determinará cómo calcular el inverso de un elemento en $\mathbb{Z}_n$:

Proposición 2.17

Sea $a \in \mathbb{Z}_n$ tal que a es invertible en $\mathbb{Z}_n$. Entonces su inverso es $a^{-1} = \frac{1 + n k}{a}$

para algún $k \in \mathbb{Z}$ tal que $0 \leq k \leq a \wedge \frac{1 + n k}{a} \in \mathbb{Z}$

Demostración

Está detallada en [4]. ■

Ejemplo 2.18

Calcular los siguientes inversos, en caso que existan, en el conjunto que se indica:

1) 6^{-1} en $\mathbb{Z}_8$

2) 5^{-1} en $\mathbb{Z}_{12}$

Solución

1) Como $\text{mcd}(6, 8) = 2 \neq 1$ por el Teorema 2.15 (Teorema de Invertibilidad) resulta que 6 no tiene inverso en $\mathbb{Z}_8$.

2) Como $\text{mcd}(5, 12) = 1$ entonces 5 tiene inverso en $\mathbb{Z}_{12}$. Por la Proposición 2.17,

$$5^{-1} = \frac{1 + 12 k}{5}$$

Para algún $k \in \mathbb{Z}$ tal que $0 \leq k \leq 5 \wedge \frac{1 + 12 k}{5} \in \mathbb{Z}$. A fin de determinar cuál es el valor de k , se diseña la siguiente tabla:

k	$\frac{1+12k}{5}$
0	$\frac{1}{5} \notin \mathbb{Z}$
1	$\frac{13}{5} \notin \mathbb{Z}$
2	$\frac{25}{5} = 5 \in \mathbb{Z}$

Por lo tanto, $5^{-1} = 5$ en $\mathbb{Z}_{12}$

Cálculo de potencias modulares

Dado $a \in \mathbb{Z}$ con $a \geq 0$ y $k \in \mathbb{N}$, el cálculo de la potencia modular consiste en hallar $a^k \pmod{n}$. Es decir, se trata de calcular el resto de la división entera entre a^k y el número n . Pero calcular a^k y luego efectuar la división entera para hallar el resto, se torna sumamente dificultoso si a es un número grande.

Además, puede producir *overflow* (desbordamiento aritmético) en una computadora.

Por esta razón presentamos el siguiente algoritmo para permitir calcular $a^k \pmod{n}$ de un modo más sencillo. El algoritmo se basa en el siguiente hecho:

Si se efectúa la división entera de k por 2, se obtiene:

$$k = 2 \left\lfloor \frac{k}{2} \right\rfloor + r$$

Donde

$\left\lfloor \frac{k}{2} \right\rfloor$: Parte entera de $\frac{k}{2}$.

r : Resto de la división ($0 \leq r \leq 1$).

En consecuencia, $a^k = a^{2 \left\lfloor \frac{k}{2} \right\rfloor + r} = \left(a^2\right)^{\left\lfloor \frac{k}{2} \right\rfloor} \cdot a^r$

Esta expresión permite generar las siguientes sucesiones:

$\{a_0, a_1, a_2, \dots\}$, $\{k_0, k_1, k_2, \dots\}$, $\{c_0, c_1, c_2, \dots\}$, $\{r_0, r_1, r_2, \dots\}$

Con

$$\begin{cases} a_0 = a \\ k_0 = k \\ c_0 = 1 \\ r_0 = r \end{cases}$$

Los demás términos de las sucesiones se definen del siguiente modo, aplicando las propiedades de congruencia enunciadas en los incisos 5), 8) y 9) de la Proposición 2.6:

$$a^k = a^{2 \left\lfloor \frac{k}{2} \right\rfloor + r_0} = \left(a^2\right)^{\left\lfloor \frac{k}{2} \right\rfloor} \cdot a^{r_0} \equiv a_1^{k_1} \cdot c_1 = \left(a_1^2\right)^{\left\lfloor \frac{k_1}{2} \right\rfloor} \cdot a_1^{r_1} \cdot c_1 \equiv a_2^{k_2} \cdot c_2 = \dots$$

$$\begin{array}{ccc} \uparrow & & \uparrow \\ \left\{ \begin{array}{l} a_1 \equiv a^2 \pmod{n} \\ k_1 = \left\lfloor \frac{k}{2} \right\rfloor \\ c_1 \equiv a^{r_0} = a^{r_0} \cdot c_0 \pmod{n} \end{array} \right. & & \left\{ \begin{array}{l} a_2 \equiv a_1^2 \pmod{n} \\ k_2 = \left\lfloor \frac{k_1}{2} \right\rfloor \\ c_2 \equiv a_1^{r_1} \cdot c_1 \pmod{n} \end{array} \right. \end{array}$$

El proceso sigue hasta que $k_i \leq 1$ para algún i . El pseudocódigo del algoritmo lo mostramos a continuación:

Algoritmo para calcular $a^k \pmod{n}$

$a_0 = a$, $k_0 = k$, $c_0 = 1$, $i = 1$

Mientras $k_{i-1} > 1$ hacer:

$a_i \equiv a_{i-1}^2 \pmod{n}$

Si k_{i-1} es par entonces

$c_i = c_{i-1}$

Sino $c_i = a_{i-1} \cdot c_{i-1} \pmod{n}$

Fin si

$k_i = \left\lfloor \frac{k_{i-1}}{2} \right\rfloor$

$i = i + 1$

Fin mientras

$a^k \equiv a_{i-1} \cdot c_{i-1} \pmod{n}$

Desarrollaremos un Ejemplo que ayudará a comprender este algoritmo.

Ejemplo 2.19

Calcular $84^{41} \pmod{247}$

Solución

En este caso, $a_0 = 84$, $c_0 = 1$ y $k_0 = 41$. A fin de ordenar los cálculos, diseñamos la siguiente tabla:

i	a_i	c_i	k_i
0	84	1	41
1	140	84	20
2	87	84	10
3	159	84	5
4	87	18	2
5	159	18	1

Los detalles de los datos que figuran en la tabla los explicamos a continuación:

$$a_1 = a_0^2 = (84)^2 = 7056 \equiv 140 \pmod{247}$$

$$c_1 = a_0 \cdot c_0 = 84 \cdot 1 = 84 \pmod{247}$$

$$k_1 = \left\lfloor \frac{k_{i-1}}{2} \right\rfloor = \left\lfloor \frac{k_0}{2} \right\rfloor = \left\lfloor \frac{41}{2} \right\rfloor = 20$$

$$a_2 = a_1^2 = (140)^2 = 19600 \equiv 87 \pmod{247}$$

$$c_2 = c_1 = 84 \pmod{247}$$

$$k_2 = \left\lfloor \frac{k_1}{2} \right\rfloor = \left\lfloor \frac{20}{2} \right\rfloor = 10$$

$$a_3 = a_2^2 = (87)^2 = 7569 \equiv 159 \pmod{247}$$

$$c_3 = c_2 = 84 \pmod{247}$$

$$k_3 = \left\lfloor \frac{k_2}{2} \right\rfloor = \left\lfloor \frac{10}{2} \right\rfloor = 5$$

$$a_4 = a_3^2 = (159)^2 = 25281 \equiv 87 \pmod{247}$$

$$c_4 = a_3 \cdot c_3 = 159 \cdot 84 = 13356 \equiv 18 \pmod{247}$$

$$k_4 = \left\lfloor \frac{k_3}{2} \right\rfloor = \left\lfloor \frac{5}{2} \right\rfloor = 2$$

$$a_5 = a_4^2 = (87)^2 = 7569 \equiv 159 \pmod{247}$$

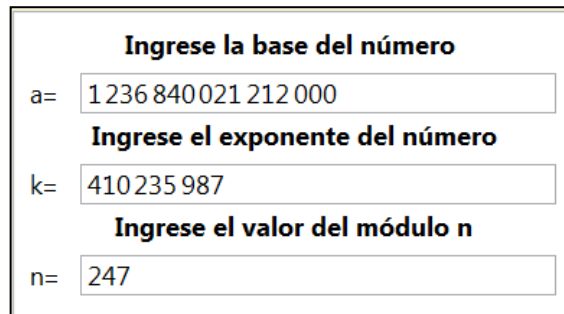
$$c_5 = c_4 = 18 \pmod{247}$$

$$k_5 = \left\lfloor \frac{k_4}{2} \right\rfloor = \left\lfloor \frac{2}{2} \right\rfloor = 1$$

Por lo tanto, $(84)^{41} = a_5 \cdot c_5 = (159) \cdot (18) = 2862 \equiv 145 \pmod{247}$

Cálculo de potencias modulares mediante Wolfram Mathematica 10.0

El siguiente programa, diseñado mediante el software Wolfram Mathematica 10.0, calcula $a^k \pmod{n}$. Se ingresan la base a , el exponente k y el módulo n , en un cuadro de diálogo similar al de la figura:



Ingrese la base del número
a= 1236 840021 212 000

Ingrese el exponente del número
k= 410235 987

Ingrese el valor del módulo n
n= 247

Figura 2.1: Ingreso por pantalla de los datos para el cálculo de la potencia modular.

Según los datos ingresados, el programa calcula

$$1236840021212000^{410235987} \pmod{247}$$

Al ejecutar el programa, se obtiene por resultado 64. El código fuente del programa lo mostramos a continuación.

Código fuente del programa que calcula $a^k \pmod{n}$

```
Interpretation[ { a = 1236840021212000, k = 410235987, n = 247 },  
Panel [ Grid[{{ Style ["Ingrese la base del número ", Bold ], SpanFromLeft },  
{"a=", InputField[ Dynamic [ a ] ] },  
{Style [ "Ingrese el exponente del número", Bold], SpanFromLeft },  
{" k =", InputField[Dynamic [ k ] ] },  
{Style ["Ingrese el valor del módulo n", Bold], SpanFromLeft},  
{"n=", InputField[ Dynamic [ n ] ] }  
} ] ],  
cc = 1;  
While[ k > 1,  
b = Mod[ a^2, n ];  
If [ Mod[ k, 2 ] != 0, cc = a*cc ];  
a = b;  
k = IntegerPart [ k/2 ];  
];  
Print["El resultado es ", Mod[a * cc, n ]  
]
```

Logaritmos discretos

Logaritmo es sinónimo de *exponente*. Cada vez que vemos una potencia, desde otro ángulo veremos un algoritmo. Por ejemplo,

$$3^4 = 81 \text{ es equivalente a decir que } \log_3 81 = 4$$

En Aritmética Modular también tenemos logaritmos, que se definen en el conjunto $\mathbb{Z}_n^* = \{1, 2, \dots, n-1\}$ siempre que el número natural n sea primo y la base del logaritmo sea “un generador de $\mathbb{Z}_n^*$ ”. Estas condiciones permiten asegurar que el logaritmo quede definido para todo elemento de $\mathbb{Z}_n^*$ (esta afirmación está demostrada en [4]). Se apellidan discretos por oposición a los continuos que son los que conocemos, y tiene que ver con la finitud del conjunto donde se definen.

El uso de la Aritmética Modular introduce una importante complejidad al problema; por esta razón el logaritmo discreto se utiliza en Criptografía.

Su importancia se debe a que la seguridad del Algoritmo de Diffie-Hellman se basa en la dificultad para resolver logaritmos discretos.

Actualmente no se conoce ningún algoritmo, para ser implementado en una computadora clásica, que permita calcular logaritmos discretos de una manera eficiente.

Antes de definirlo matemáticamente, introduciremos el siguiente concepto:

Definición 2.20: Elemento generador de $\mathbb{Z}_n^*$

Sea $a \in \mathbb{Z}_n^* = \{1, 2, \dots, n-1\}$. Diremos que a es un generador de $\mathbb{Z}_n^*$ si verifica que

$$\mathbb{Z}_n^* = \{a^0, a^1, \dots, a^{n-2}\}$$

Definición 2.21: Logaritmo discreto en $\mathbb{Z}_n^*$

Sea n un número primo y $\mathbb{Z}_n^* = \{1, 2, \dots, n-1\}$. Si $a \in \mathbb{Z}_n^*$ es un generador de $\mathbb{Z}_n^*$, el logaritmo discreto en base “ a ” de un número $x \in \mathbb{Z}_n^*$ es un número $y \in \mathbb{Z}_n$ que verifica:

$$\log_a(x) = y \Leftrightarrow a^y = x \quad (\text{en } \mathbb{Z}_n)$$

El siguiente Ejemplo aclarará estas definiciones.

Ejemplo 2.22

Sea $\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$ con la operación producto en $\mathbb{Z}_7^*$.

- 1) Probar que el número 3 es un generador de $\mathbb{Z}_7^*$.
- 2) Calcular $\log_3(x) \forall x \in \mathbb{Z}_7^*$.
- 3) Probar que el número 2 no es un generador de $\mathbb{Z}_7^*$.
- 4) Verificar que $\log_2(x)$ no queda definido $\forall x \in \mathbb{Z}_7^*$.

Solución

- 1) En este caso, $n = 7$. Verificaremos que

$$\{3^0, 3^1, 3^2, 3^3, 3^4, 3^5\} = \mathbb{Z}_7^*$$

Para ello diseñamos la siguiente tabla:

i	0	1	2	3	4	5
3^i	1	3	2	6	4	5

Por lo tanto, las potencias de 3 generan todos los elementos de $\mathbb{Z}_7^*$.

- 2) Como $n = 7$ es un número primo y las potencias de 3 generan todos los elementos de $\mathbb{Z}_7^*$, $\log_3(x)$ queda definido $\forall x \in \mathbb{Z}_7^*$ del siguiente modo:

$$\log_3(1) = 0 \text{ pues } 3^0 = 1 \text{ en } \mathbb{Z}_7^*$$

$$\log_3(2) = 2 \text{ pues } 3^2 = 9 = 2 \text{ en } \mathbb{Z}_7^*$$

$$\log_3(3) = 1 \text{ pues } 3^1 = 3 \text{ en } \mathbb{Z}_7^*$$

$$\log_3(4) = 4 \text{ pues } 3^4 = 81 = 4 \text{ en } \mathbb{Z}_7^*$$

$$\log_3(5) = 5 \text{ pues } 3^5 = 243 = 5 \text{ en } \mathbb{Z}_7^*$$

$$\log_3(6) = 3 \text{ pues } 3^3 = 27 = 6 \text{ en } \mathbb{Z}_7^*$$

- 3) Verificaremos que 2 no es generador de $\mathbb{Z}_7^*$:

i	0	1	2	3	4	5
2^i	1	2	4	1	2	4

Como $\{2^0, 2^1, 2^2, 2^3, 2^4, 2^5\} = \{1, 2, 4\} \neq \mathbb{Z}_7^*$, queda probado que 2 no es generador de $\mathbb{Z}_7^*$.

4) Como 2 no es generador de $\mathbb{Z}_7^*$, $\log_2(x)$ no queda definido $\forall x \in \mathbb{Z}_7^*$. Por ejemplo, $\log_2(3)$ no existe ya que no existe $y \in \mathbb{Z}_7$ que verifica que $2^y = 3$ (en $\mathbb{Z}_7$). Ello se debe a que $3 \notin \{2^y / y \in \mathbb{Z}_7\} = \{1, 2, 4\}$.

Observación 2.23

Cabe plantearnos lo siguiente: Si n es un número primo, ¿existirá algún $a \in \mathbb{Z}_n^*$ que sea generador de $\mathbb{Z}_n^*$? La respuesta es que sí existe, y la prueba de esta afirmación está desarrollada en [4].

El problema de los logaritmos discretos

Nos preguntamos si existe algún algoritmo que permita calcular logaritmos discretos de una manera más sencilla, sin hacerlos “a fuerza bruta”. La respuesta es NO. Se tiene que hacer por exhaustión, y mientras más grande sea el módulo n , mayor será el tiempo que tome hallar el logaritmo. La dificultad del cálculo de los logaritmos discretos es lo que da seguridad al algoritmo de Diffie-Hellman.

Almacenamiento de datos en un computador

De los numerosos sistemas de numeración posibles, el más sencillo de todos es el sistema binario. En dicho sistema sólo se necesitan dos símbolos: 0 y 1.

Así, una determinada cantidad puede representarse mediante una sucesión de unos y ceros, los cuales simbolizan dos estados: encendido-apagado o mayor voltaje-menor voltaje. De este modo, el computador utiliza el sistema binario para almacenar los datos. La unidad mínima de almacenamiento es el *bit* (**binary digit**) que puede guardar 1 o bien 0. Se puede imaginar un *bit* como una bombilla que puede estar en uno de los dos estados: apagado (estado 0) o encendido (estado 1).

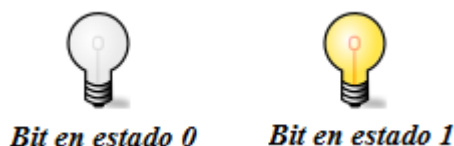


Figura 2.2: Representación gráfica del bit.
Fuente: <https://es.wikipedia.org/wiki/Bit>

O alternativamente, al *bit* se lo representa mediante un casillero, para indicar un espacio de memoria en el que puede albergar un 1 o un 0.



Figura 2.3: Representación gráfica del bit mediante un casillero.

La unidad mínima de almacenamiento de datos: El Byte.

El computador, como sabemos, debe ser capaz de almacenar cantidades numéricas y de operar con ellas. A los efectos de poder cumplir con este objetivo, introduciremos el concepto de *Byte*.

El *byte* es un conjunto formado por 8 bits y constituye la unidad mínima de almacenamiento de datos.

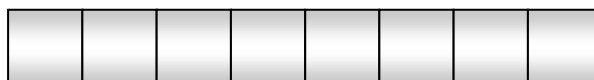


Figura 2.4: Representación gráfica del byte mediante 8 casilleros.

El siguiente ejemplo ilustrará cuál es el mayor número entero que se puede almacenar en 1 byte y cuántos bytes se necesitan para operar con dichos números enteros.

Ejemplo 2.24

- 1) Determinar el mayor número entero que se puede almacenar en 1 byte.
- 2) Determinar cuántos bytes se necesitan para almacenar el resultado de la suma y de la multiplicación de dos enteros no negativos que ocupen 1 byte cada uno.

Solución

- 1) El mayor número entero es aquél que tiene “unos” en todos los bits que componen el byte. Es decir, es el número binario

1 1 1 1 1 1 1 1

Este número expresado en base 10 es

$$\sum_{i=0}^7 1 \cdot 2^i$$

Teniendo en cuenta que la fórmula de una suma geométrica de razón $q \in \mathbb{R}$ es

$$\sum_{i=0}^n q^i = \frac{1 - q^{n+1}}{1 - q}$$

Entonces, tomando $q = 2$ resulta

$$\sum_{i=0}^7 2^i = \frac{1 - 2^8}{1 - 2} = 255$$

Por lo tanto, 255 es el mayor entero que se puede almacenar en un byte.

- 2) Sean a y b dos enteros no negativos almacenados cada uno de ellos en un byte. Supondremos que $a = b = 255$, ya que estos son los mayores números enteros que se pueden guardar en un byte.

Para $a + b$:

$a + b = 255 + 255 = 510$. Dicho número expresado en base 2 es:

1 1 1 1 1 1 1 0

Por lo tanto, necesitamos 9 bits para almacenar $a + b$. En consecuencia, como la unidad mínima de almacenamiento de datos es el byte, precisaremos 2 bytes.

Para $a.b$:

$a.b = (255).(255) = 65025$. Este número expresado en base 2 es

1 1 1 1 1 1 1 0 0 0 0 0 0 0 1

Luego, necesitamos 16 bits para almacenar $a.b$. Es decir, 2 bytes.

En consecuencia, precisaremos 2 bytes para poder multiplicar y sumar números enteros no negativos que ocupan un byte.

Orden de complejidad computacional de un algoritmo

Los avances en Computación de la segunda mitad del siglo XX pusieron en riesgo la integridad de los sistemas criptográficos usados hasta entonces, a la vez que proporcionaban medios para la creación de otros nuevos y más seguros. Es en este punto donde la Teoría de la Complejidad, encargada de estudiar los recursos necesarios para el cómputo de los algoritmos, cobra importancia dentro de la Criptografía. Esto es debido a que los nuevos algoritmos criptográficos basarán su seguridad en la *dificultad* para ejecutar las operaciones que permitan obtener la información original de un criptograma. De este modo, la ya mencionada Teoría de Complejidad será quien determinará cuál es esta *dificultad*.

Por otro lado, desde el punto de vista de la Ingeniería y de la Informática, lo que preocupa son los recursos físicos necesarios para que un algoritmo pueda ejecutarse. De todos ellos, los más importantes son dos: **el tiempo de ejecución** y **la cantidad de memoria** que requiere el algoritmo. Estos dos parámetros definen básicamente el orden de complejidad computacional de un algoritmo y, para medirlos, se utiliza el concepto de “funciones de orden O ”. Su definición la daremos a continuación.

Definición 2.25: Funciones de orden O

Sea $f: \mathbb{N} \rightarrow [0, +\infty)$. El conjunto de funciones de orden O de f , simbolizado por $O(f)$, es

$$O(f) = \left\{ g: \mathbb{N} \rightarrow \mathbb{R} \text{ tal que } \exists k > 0 \wedge n_0 \in \mathbb{N} / |g(n)| \leq k \cdot f(n) \quad \forall n \geq n_0 \right\}$$

Así, una función $g: \mathbb{N} \rightarrow \mathbb{R}$ se dice que es $O(f)$ si $g \in O(f)$.

La siguiente Proposición demuestra algunas propiedades de las funciones de orden O .

Proposición 2.26

- 1) Si $\lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} \leq M$ para alguna constante M entonces $g \in O(f)$.
- 2) Si $f_1 \in O(g)$ y $f_2 \in O(h) \Rightarrow f_1 \cdot f_2 \in O(g \cdot h)$
- 3) $f \in O(f)$
- 4) $O(f) = O(g) \Leftrightarrow f \in O(g)$ y $g \in O(f)$
- 5) Si $f_1 \in O(g) \wedge f_2 \in O(h) \Rightarrow f_1 + f_2 \in O(\max(g, h))$

Demostración

1) Sea $L = \lim_{n \rightarrow \infty} \frac{g(n)}{f(n)}$. Entonces, por definición de límite, se tiene que

$$\forall \varepsilon > 0 \exists n_0 / \forall n \geq n_0 \text{ se verifica que } \left| \frac{g(n)}{f(n)} - L \right| < \varepsilon$$

En consecuencia, $\forall n \geq n_0$ resulta

$$-\varepsilon < \frac{g(n)}{f(n)} - L < \varepsilon \Rightarrow L - \varepsilon < \frac{g(n)}{f(n)} < \varepsilon + L. \text{ Como } -L - \varepsilon < L - \varepsilon \text{ entonces}$$

$$-L - \varepsilon < \frac{g(n)}{f(n)} < \varepsilon + L$$

Por lo tanto

$$\left| \frac{g(n)}{f(n)} \right| < \varepsilon + L \quad \forall n \geq n_0$$

Teniendo en cuenta que $L < M$ se tiene

$$\left| \frac{g(n)}{f(n)} \right| < \varepsilon + M \quad \forall n \geq n_0$$

Tomando $k = \varepsilon + M$ y teniendo en cuenta que $f(n) \geq 0$, se obtiene

$$|g(n)| < k f(n) \quad \forall n \geq n_0$$

Luego, $g \in O(f)$.

2) Como $f_1 \in O(g)$ y $f_2 \in O(h)$ resulta

$$\exists k_1 > 0 \wedge n_1 \in \mathbb{N} / |f_1(n)| \leq k_1 \cdot g(n) \quad \forall n \geq n_1 \text{ y además}$$

$$\exists k_2 > 0 \wedge n_2 \in \mathbb{N} / |f_2(n)| \leq k_2 \cdot h(n) \quad \forall n \geq n_2$$

Tomando $k = k_1 \cdot k_2$ y $n_0 = \max\{n_1, n_2\}$ entonces $\forall n \geq n_0$ se verifica que

$$\frac{|f_1(n) \cdot f_2(n)|}{g(n) h(n)} = \frac{|f_1(n)|}{g(n)} \cdot \frac{|f_2(n)|}{h(n)} \leq k_1 \cdot k_2 = k. \text{ Por lo tanto,}$$

$$|f_1(n) \cdot f_2(n)| \leq k g(n) \cdot h(n) \quad \forall n \geq n_0$$

Luego $f_1 \cdot f_2 \in O(g \cdot h)$

3) Teniendo en cuenta que $f(n) \geq 0$ y considerando $k=1$ y $n_0=1$, resulta

$$f(n) \leq 1 \cdot f(n) \quad \forall n \geq 1$$

Por lo tanto, $f \in O(f)$

4) $\Rightarrow$ Por el inciso 3), $f \in O(f)$. Como $O(f) = O(g)$ entonces $f \in O(g)$.

De manera similar se prueba que si $g \in O(g)$ entonces $g \in O(f)$.

$\Leftarrow$ Sea $h \in O(f)$. Probemos que $h \in O(g)$. Como

$$h \in O(f), \exists k_1 > 0 \wedge n_1 \in \mathbb{N} \text{ tal que } |h(n)| \leq k_1 f(n) \quad \forall n \geq n_1$$

Por otro lado, como

$$f \in O(g), \exists k_2 > 0 \wedge n_2 \in \mathbb{N} \text{ tal que } |f(n)| \leq k_2 g(n) \quad \forall n \geq n_2$$

Sea $n_0 = \max\{n_1, n_2\}$ y $k = k_1 k_2$. Teniendo en cuenta que $f(n) \geq 0 \wedge g(n) \geq 0$, si $n \geq n_0$ se verifica que

$$|h(n)| \leq k_1 f(n) = k_1 |f(n)| \leq k_1 k_2 g(n) = k g(n). \text{ Por lo tanto } h \in O(g)$$

De manera similar, se prueba que si $h \in O(g)$ entonces $h \in O(f)$.

5) Como $f_1 \in O(g)$, $\exists k_1 > 0 \wedge n_1 \in \mathbb{N}$ tal que $|f_1(n)| \leq k_1 g(n) \quad \forall n \geq n_1$.

Por otro lado, como

$$f_2 \in O(h), \exists k_2 > 0 \wedge n_2 \in \mathbb{N} \text{ tal que } f_2(n) \leq k_2 h(n) \quad \forall n \geq n_2$$

Por lo tanto, tomando $n_0 = \max\{n_1, n_2\}$ y $k = k_1 + k_2$ resulta que $\forall n \geq n_0$ se verifica que

$$\begin{aligned} |f_1(n) + f_2(n)| &\leq |f_1(n)| + |f_2(n)| \leq k_1 g(n) + k_2 h(n) \leq \\ &\leq k_1 \max(g(n), h(n)) + k_2 \max(g(n), h(n)) \leq (k_1 + k_2) \max(g(n), h(n)) = \\ &= k \max(g(n), h(n)) \end{aligned}$$

En consecuencia, $f_1 + f_2 \in O(\max(g, h))$ ■

Ejemplo 2.27

Si $g(n) = 5 \cdot 2^n + n^3 + 4$ y $f(n) = 2^n$ probar que $g \in O(f)$

Solución

Aplicaremos el inciso 1) de la Proposición 2.24

$$\lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = \lim_{n \rightarrow \infty} \frac{5 \cdot 2^n + n^3 + 4}{2^n} = \lim_{n \rightarrow \infty} 5 + \frac{n^3}{2^n} + \frac{4}{2^n}$$

Si consideramos $x \in \mathbb{R}$, y aplicando sucesivas veces la Regla de L'Hopital, resulta

$$\lim_{x \rightarrow \infty} \frac{x^3}{2^x} = \lim_{x \rightarrow \infty} \frac{3x^2}{2^x \ln(2)} = \lim_{x \rightarrow \infty} \frac{6x}{2^x (\ln(2))^2} = \lim_{x \rightarrow \infty} \frac{6}{2^x (\ln(2))^3} = 0$$

En consecuencia, si $n \in \mathbb{N}$ se obtiene que $\lim_{n \rightarrow \infty} \frac{n^3}{2^n} = 0$. Por otro lado, $\lim_{n \rightarrow \infty} \frac{4}{2^n} = 0$

$$\text{Luego, } \lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = 5$$

Tomando $M = 2$ resulta entonces que $g \in O(f)$

Observación 2.28

A menudo, para enfatizar que las funciones f y g tienen por variable independiente a un número natural n , en lugar de afirmar que g es $O(f)$ se suele decir que $g(n)$ es $O(f(n))$. Esta notación es la que se aplicará en adelante.

A continuación, definiremos el orden de complejidad computacional de un algoritmo, concepto que aplicaremos en el Capítulo 5.

Definición 2.29: Orden de complejidad computacional de un algoritmo

Se dice el orden de complejidad de un algoritmo es $O(f(n))$ donde $f: \mathbb{N} \rightarrow [0, +\infty)$ si el tiempo de ejecución o la cantidad de memoria requerida para ejecutar el algoritmo se expresa mediante una función $g(n)$, con $g(n) \in O(f(n))$

En este caso, “ n ” puede representar la longitud de un vector, el orden de una matriz, el número de registros de una base de datos, etc. Por otro lado, la definición de $O(f)$ permite asegurar que, a partir de cierto “ n ” en adelante, el tiempo de ejecución de un algoritmo o la cantidad de memoria requerida nunca será superior a $f(n)$. Es decir que, si “ n ” es suficientemente grande, y salvo una constante multiplicativa “ k ”, $f(n)$ es una

cota superior de $g(n)$. Además, el hecho de indicar, en la definición de $O(f)$, que “ n ” sea mayor que cierto valor n_0 se debe a que la eficiencia de un algoritmo se mide para valores de “ n ” muy grandes. Ya que, casi siempre, los problemas pequeños se pueden resolver de cualquier manera, apareciendo las limitaciones al atacar problemas grandes. Por lo general, la función $f(n)$ se escoge de modo tal que sea lo más sencilla posible. Esto da lugar a la siguiente definición.

Definición 2.30: Clasificación del orden de complejidad de un algoritmo

- 1) Si $f(n) = \log(n)$ se dice que el algoritmo es de orden logarítmico.
- 2) Si $f(n) = n^p$ con $p \in \mathbb{N}$ se dice que el algoritmo es de orden polinomial.
En este caso, se lo considera **eficiente**.
- 3) Si $f(n) = a^n$ con $a > 1$, se dice que el algoritmo es de orden exponencial o No Polinomial (NP). En este caso, se lo considera **ineficiente**.

Los algoritmos de **orden logarítmico** son los ideales, ya que el tiempo que demandan para procesar los datos se reduce considerablemente.

Los de **orden polinomial** se enfrentan a una dificultad a medida que “ p ” crece. Mientras menor sea el valor de “ p ”, más eficiente será el algoritmo. Complejidades del orden $O(n^2)$ y $O(n^2)$ suelen ser efectivamente abordables, y prácticamente nadie acepta algoritmos de orden $O(n^{100})$.

Los algoritmos de **orden exponencial** suelen ser intratables y sólo son aplicables a problemas con valores de “ n ” pequeños.

A la vista de lo expuesto, se comprende que los programadores busquen algoritmos de orden polinomial, ya que es un golpe de suerte encontrar de orden logarítmico.

Se finalizará este capítulo mostrando un ejemplo que ilustrará cómo clasificar el orden de complejidad de un algoritmo.

Estructuras algebraicas

A continuación, definiremos las estructuras algebraicas que aplicaremos en el Capítulo 5.

Definición 2.31: Semigrupo

Sea G un conjunto no vacío y $*$ una operación binaria definida sobre los elementos de G . El par $(G, *)$ es un semigrupo si y sólo si verifica las siguientes propiedades:

- 1) $*$ es Ley de Composición Interna: Si $a, b \in G \Rightarrow a * b \in G$.
- 2) Propiedad asociativa: Si $a, b, c \in G \Rightarrow (a * b) * c = a(b * c)$.

Definición 2.32: Grupo

Sea G un conjunto no vacío y $*$ una operación binaria definida sobre los elementos de G . El par $(G, *)$ es un grupo si y sólo si verifica las siguientes propiedades:

- 1) $*$ es Ley de Composición Interna: Si $a, b \in G \Rightarrow a * b \in G$.
- 2) Propiedad asociativa: Si $a, b, c \in G \Rightarrow (a * b) * c = a(b * c)$.
- 3) Existencia de elemento neutro o Identidad: $\exists e \in G / a * e = e * a \quad \forall a \in G$.
- 3) Existencia de inversos: $\forall a \in G \exists a^{-1} \in G / a * a^{-1} = a^{-1} * a = e$.

Definición 2.33: Grupo Abelian o conmutativo

Un grupo $(G, *)$ es abeliano si verifica la Propiedad Conmutativa:

$$\text{Si } a, b \in G \Rightarrow a * b = b * a.$$

Definición 2.34: Anillo

Sea A un conjunto no vacío con 2 leyes de composición interna $*$ y $\bullet$, la terna $(A, *, \bullet)$ es un anillo si:

- 1) $(A, *)$ es un grupo abeliano
- 2) $(A, \bullet)$ es un semigrupo
- 3) $\bullet$ es doblemente distributiva con respecto a $*$, es decir:

$$a \bullet (b * c) = (a \bullet b) * (a \bullet c)$$

$$(b * c) \bullet a = (b \bullet a) * (c \bullet a) \quad \forall a, b, c \in A$$

Nota: Si la operación $\bullet$ es conmutativa, se dice que el Anillo es conmutativo. Si existe elemento neutro con respecto a la operación $\bullet$, que lo simbolizaremos por 1 , se dice que $(A, *, \bullet)$ es un anillo con identidad.

Representaremos al anillo $(A, *, \bullet)$ por $(A, +, \cdot)$ y a su elemento neutro con respecto a la operación $+$, lo denotaremos por 0 .

Ejemplo 2.35

- 1) El conjunto de matrices de orden n a coeficientes reales, es decir,

$$M_n(\mathbb{R}) = \left\{ A = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{n1} & \dots & a_{nn} \end{pmatrix} \text{ tal que } a_{ij} \in \mathbb{R} \quad \forall i, j = 1, 2, \dots, n \right\}$$

Es un Semigrupo bajo la operación producto matricial.

- 2) El conjunto de matrices invertibles de orden n a coeficientes reales, es decir,

$$I_n(\mathbb{R}) = \left\{ A = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{n1} & \dots & a_{nn} \end{pmatrix} \text{ tal que } \det(A) \neq 0 \wedge a_{ij} \in \mathbb{R} \quad \forall i, j = 1, 2, \dots, n \right\}$$

Es un Grupo no abeliano bajo la operación producto matricial.

- 3) Si n es un número primo entonces el par $(\mathbb{Z}_n, \bullet)$, es un grupo abeliano, donde $\bullet$ es el producto en $\mathbb{Z}_n$.

- 4) Si $n \in \mathbb{N}$, la terna $(\mathbb{Z}_n, +, \bullet)$ es un anillo conmutativo donde $+$ y $\bullet$ son, respectivamente, la suma y producto en $\mathbb{Z}_n$.

- 5) La terna $(M_n(\mathbb{R}), +, \cdot)$ es un anillo no conmutativo donde $+$ y $\cdot$ son, respectivamente, la suma y producto matricial.

Capítulo 3

Primeros métodos criptográficos

Capítulo 3: Primeros métodos criptográficos

En este capítulo describiremos dos de los métodos criptográficos que adquirieron históricamente gran renombre: la **Escítala** y el **Método de Matrices y Digrafías**.

La Escítala

La Escítala fue el primer sistema criptográfico de la historia. Fue utilizada en el siglo V antes de Cristo por los militares espartanos, a fin de proteger el contenido de sus mensajes de los ojos de sus enemigos.



Figura 3.1: La Escítala. Fuente: <http://ojoscuriosos.com/la-escitala/>.

Está formada por dos palos de madera de igual diámetro (un palo para el emisor del mensaje y el otro palo para el receptor) y de una cinta de cuero o papiro sobre la cual se escribe el mensaje. La escítala funciona del siguiente modo:

Emisor del mensaje:

- 1) Enrolla la cinta en forma de espiral sobre el palo de madera.
- 2) Escribe el mensaje horizontalmente, de izquierda a derecha, de forma tal que en cada vuelta de cinta aparezca cada vez una letra.
- 3) Desenrolla la cinta, quedando escrito el mensaje con las letras desordenadas.
- 4) Envía la cinta con el mensaje encriptado al receptor.

Receptor del mensaje:

- 1) Enrolla la cinta recibida sobre su palo de madera y simplemente lee el mensaje.

Este método es considerado el primer sistema criptográfico **por transposición**, el cual consiste en ocultar el significado real de un texto alterando el orden de los símbolos que lo conforman.

Si bien en la antigüedad resultó ser un método seguro para ocultar información, en la actualidad es muy primitivo. Podemos diseñar un algoritmo para ser implementado en una computadora clásica, que permita descubrir rápidamente el mensaje cifrado. A continuación veremos cómo hacerlo.

Desencriptado de la escítala mediante Wolfram Mathematica 10.0

El siguiente programa, diseñado mediante el software Wolfram Mathematica 10.0, permite descifrar un mensaje encriptado con la escítala. El mensaje se ingresa tal cual se lee en la cinta desenrollada, en un cuadro de diálogo como se muestra en la figura:



Figura 3.2: Ingreso por pantalla del mensaje cifrado.

El programa desconoce la cantidad de vueltas dadas a la cinta, ya que supone que un intruso capturó el mensaje cifrado y no sabe cuál es el diámetro de los palos utilizados para cifrar el mensaje. Por esta razón el software intenta descifrarlo probando hacerlo con dos, tres y hasta $\left\lceil \frac{n}{2} \right\rceil$ vueltas de cinta, donde n es la cantidad de símbolos que tiene el mensaje. El texto ingresado se descifra para cada una de estas posibles vueltas. De este modo, aquél que muestre un “texto coherente” será el mensaje que envió el emisor. La siguiente figura muestra los posibles textos desencriptados, que corresponde al texto cifrado: **ATCFORIGIARP**. En este caso, $n = 12$, con lo cual se intenta descifrar el mensaje probando con 2, 3, 4, 5 y $\left\lceil \frac{n}{2} \right\rceil = \left\lceil \frac{12}{2} \right\rceil = 6$ vueltas de cinta.

Número de vueltas: 2	Número de vueltas: 3	Número de vueltas: 4																																				
Posible texto desencriptado:	Posible texto desencriptado:	Posible texto desencriptado:																																				
<table><tr><td>R</td><td>P</td></tr><tr><td>O</td><td>R</td></tr><tr><td>F</td><td>A</td></tr><tr><td>C</td><td>I</td></tr><tr><td>T</td><td>G</td></tr><tr><td>A</td><td>I</td></tr></table>	R	P	O	R	F	A	C	I	T	G	A	I	<table><tr><td>F</td><td>G</td><td>P</td></tr><tr><td>C</td><td>I</td><td>R</td></tr><tr><td>T</td><td>R</td><td>A</td></tr><tr><td>A</td><td>O</td><td>I</td></tr></table>	F	G	P	C	I	R	T	R	A	A	O	I	<table><tr><td>C</td><td>R</td><td>I</td><td>P</td></tr><tr><td>T</td><td>O</td><td>G</td><td>R</td></tr><tr><td>A</td><td>F</td><td>I</td><td>A</td></tr></table>	C	R	I	P	T	O	G	R	A	F	I	A
R	P																																					
O	R																																					
F	A																																					
C	I																																					
T	G																																					
A	I																																					
F	G	P																																				
C	I	R																																				
T	R	A																																				
A	O	I																																				
C	R	I	P																																			
T	O	G	R																																			
A	F	I	A																																			
Número de vueltas: 5	Número de vueltas: 6																																					
Posible texto desencriptado:	Posible texto desencriptado:																																					
<table><tr><td>C</td><td>R</td><td>G</td><td>A</td><td>P</td></tr><tr><td>T</td><td>O</td><td>I</td><td>I</td><td>R</td></tr><tr><td>A</td><td>F</td><td></td><td></td><td></td></tr></table>	C	R	G	A	P	T	O	I	I	R	A	F				<table><tr><td>T</td><td>F</td><td>R</td><td>G</td><td>A</td><td>P</td></tr><tr><td>A</td><td>C</td><td>O</td><td>I</td><td>I</td><td>R</td></tr></table>	T	F	R	G	A	P	A	C	O	I	I	R										
C	R	G	A	P																																		
T	O	I	I	R																																		
A	F																																					
T	F	R	G	A	P																																	
A	C	O	I	I	R																																	

Figura 3.3: Posibles textos desencriptados.

Al observar en la figura anterior los resultados obtenidos, concluimos que el emisor ha efectuado 4 vueltas de cinta y que el mensaje original es, leyendo de izquierda a derecha y a partir de la primera fila: **CRITOGRAFIA**.

Código fuente del programa que descripta un mensaje cifrado con la escítala

```
Interpretation[ { Mensajec = "ATCFORIGIARP"},
  Panel[ Grid[ { { Style [ "Ingrese el mensaje a descriptar:", Bold],
    SpanFromLeft },
    { Style [ "Los caracteres del mensaje se ingresan de abajo hacia arriba:",
      Bold ], SpanFromLeft }, { "Mensaje (entre comillas y sin dejar espacios en
    blanco):", InputField [ Dynamic [ Mensajec ] ] } } ],
  n = Length [ Characters [ Mensajec ] ];
  For [ vueltas = 2, vueltas <= IntegerPart [ n/2 ], vueltas ++,
    If [ Mod [ n, vueltas ] != 0,
      Filas = IntegerPart [ n/vueltas ] + 1, Filas = n/vueltas ];
  Matriz = { };
  For [ i = 1, i <= Filas, i++,
    F = { };
    For [ j = 1, j <= vueltas, j++,
      F = Append[ F, " " ]
    ];
    Matriz = Append[ Matriz, F ]
  ];
  j = 1;
  contadorFila = Filas;
  For [ i = 1, i <= n, i++,
    If [ contadorFila < 1,
      contadorFila = Filas;
      j = j + 1;
    ];
    If [ Mod[ n, vueltas ] != 0,
      If [ ( i > Filas * Mod [ n, vueltas ] ) && (contadorFila == Filas),
        Matriz[ [ contadorFila, j ] ] = " ";
        contadorFila = contadorFila - 1
      ]
    ];
    Matriz[ [ contadorFila, j ] ] = Part [ Characters [ Mensajec ], i ];
    contadorFila = contadorFila - 1;

  ];
  Print [ "Número de vueltas: ", vueltas];
  Print [ " Posible texto descriptado: "];
  Print [ Grid[Matriz, Frame -> All ] ]
]
```

El Método de Matrices y Digrafas.

Como vimos en el caso de la escítala, mucho antes de la computación electrónica se hacía Criptografía con métodos que, para la época, eran bastante seguros.

Sin embargo, con la llegada de los ordenadores y al aumentar la comunicación entre distintas regiones del mundo, hubo necesidad de crear métodos de cifrado más complejos basados en algoritmos matemáticos. Inicialmente estos algoritmos utilizaban Criptografía Simétrica, es decir, empleaban la misma clave para encriptar que para desencriptar. Los primeros fueron el **Método de Julio César** y el **Método Lineal o de Julio César Mejorado**. Ambos se encuentran detalladamente descriptos en [4]. El método de **Matrices y Digrafas** es una variante más sofisticada que el de **Julio César Mejorado**, razón por la cual brinda mayor seguridad. El mismo se basa, al igual que los dos métodos anteriores, en la codificación numérica del alfabeto, tema que expondremos a continuación.

Codificación del alfabeto

En Criptografía, cada letra del alfabeto se puede codificar con dos dígitos según la siguiente tabla, llamada Tabla de Equivalencia Numérica (TEN).

A	B	C	D	E	F	G	H	I	J	K	L	M	N
01	02	03	04	05	06	07	08	09	10	11	12	13	14

Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z	
15	16	17	18	19	20	21	22	23	24	25	26	00	

Figura 3.4: Tabla de Equivalencia Numérica (TEN)

Vulnerabilidad del Método de Julio César Mejorado

En la lengua española, las letras más usadas son en primer lugar la “E” y en segundo lugar la “A”. Este hecho facilita notablemente el criptoanálisis. Así por ejemplo, en el Método de Julio César Mejorado, podrían fácilmente descubrirse las claves “a” y “b” que conforman la función de desencriptado

$$f(x) = a \cdot x + b \pmod{27}$$

El siguiente Ejemplo nos muestra cómo, con esta información, se puede hallar rápidamente esta función.

Ejemplo 3.1

Un espía secuestró un mensaje cifrado con el Método de Julio César Mejorado. Pudo detectar que la letra que figura con mayor frecuencia es la Q y, en segundo lugar, la M. Hallar la función lineal que permita desencriptar el mensaje.

Solución

La función que descripta el mensaje es de la forma

$$f(x) = a \cdot x + b \pmod{27}$$

Calcularemos a y b .

Diseñamos la siguiente tabla, teniendo en cuenta que las letras que figuran con mayor frecuencia son la E y la A :

Letra cifrada	Código TEN de la letra cifrada	Letra descifrada	Código TEN de la letra descifrada
Q	18	E	05
M	13	A	01

De aquí deducimos que $f(x)$ verifica que

$$\begin{cases} f(18) = 5 \pmod{27} \\ f(13) = 1 \pmod{27} \end{cases} \Rightarrow \begin{cases} a \cdot 18 + b = 5 \pmod{27} & [I] \\ a \cdot 13 + b = 1 \pmod{27} & [II] \end{cases}$$

De la ecuación [II] despejamos b :

$$b = 1 - 13 \cdot a \pmod{27}$$

Reemplazando esta última ecuación en [I] y teniendo en cuenta la Proposición 2.17 resulta

$$\begin{aligned} a \cdot 18 + 1 - 13 \cdot a &= 5 \pmod{27} \\ 5 \cdot a + 1 &= 5 \pmod{27} \\ 5 \cdot a &= 4 \pmod{27} \\ a &= 5^{-1} \cdot 4 \pmod{27} \\ a &= (11) \cdot 4 \pmod{27} \\ a &= 44 \pmod{27} \\ a &= 17 \pmod{27} \end{aligned}$$

Por lo tanto, aplicando la Proposición 2.6 inciso 11) se obtiene

$$\begin{aligned} b &= 1 - 13 \cdot (17) \pmod{27} \\ b &= -220 \pmod{27} \\ b &= 23 \pmod{27} \end{aligned}$$

En consecuencia, la función a descryptar es

$$f(x) = 17 \cdot x + 23 \pmod{27}$$

Descripción del Método de Matrices y Digrafas

Como vimos en el Ejemplo anterior, el análisis de frecuencias con letras tan abundantes como la E y la A ayuda enormemente al criptoanálisis, lo cual no es deseable para el criptógrafo. Una forma de evitar esta debilidad es usar pares de letras o digrafías (doble grafía) como unidades básicas del mensaje. En base a esta idea, el Método de Matrices y Digrafías consiste en encriptar las letras del mensaje tomándolas de a pares. De este modo, la función a encriptar es

$$f(x, y) = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \pmod{27}$$

Donde

$$a, b, c, d \in \mathbb{Z}_{27}$$

x, y : Son los números que le corresponden, en la tabla TEN, a un par de letras del mensaje.

En este caso, la clave para encriptar es la matriz $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$. Para desencriptar el mensaje la matriz A debe ser invertible, es decir, debe existir $(\det(A))^{-1}$ en $\mathbb{Z}_{27}$, con lo cual

$$A^{-1} = \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} \cdot (\det(A))^{-1} \pmod{27}$$

Así, la función que desencripta es

$$g(x, y) = A^{-1} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \pmod{27}$$

Este método pertenece a la rama de la Criptografía Simétrica, ya que utiliza la misma clave para encriptar (la matriz A) que para desencriptar (la matriz A^{-1} , que se calcula en base a la matriz A)

Ejemplo 3.2

Aplicar el Método de Matrices y Digrafías para encriptar el mensaje VASO, donde la función a encriptar es

$$f(x, y) = \begin{pmatrix} 7 & 3 \\ 5 & 2 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \pmod{27}$$

Solución

Encriptaremos cada par de caracteres que componen el mensaje. Es decir, cifraremos VA y SO.

Para ello elaboramos la siguiente tabla:

Digrafiya del mensaje a cifrar	Código numérico de la digrafiya	$f(x, y)$	Código alfabético de $f(x, y)$
VA	$(x, y) = (23, 01)$	$(02, 09)$	BI
SO	$(x, y) = (20, 16)$	$(26, 24)$	YW

Para explicar cómo se obtuvieron los datos de la tabla, desarrollaremos cada fila de la misma:

Primera Fila: Encriptación de VA

A cada letra que compone la digrafiya VA le hacemos corresponder su código numérico en la tabla TEN. De este modo, a la digrafiya VA le corresponde el par ordenado $(23, 01)$

A continuación calculamos

$$f(23, 01) = \begin{pmatrix} 7 & 3 \\ 5 & 2 \end{pmatrix} \cdot \begin{pmatrix} 23 \\ 01 \end{pmatrix} \pmod{27} = \begin{pmatrix} 7 \cdot (23) + 3 \cdot (01) \\ 5 \cdot (23) + 2 \cdot (01) \end{pmatrix} \pmod{27} =$$

$$= \begin{pmatrix} 162 \\ 117 \end{pmatrix} \pmod{27} = \begin{pmatrix} 02 \\ 09 \end{pmatrix} \pmod{27}$$

Por lo tanto

$$f(23, 01) = (02, 09) \pmod{27}$$

Finalmente, al par ordenado $(02, 09)$ le corresponde, en la tabla TEN, a la digrafiya BI.

Segunda Fila: Encriptación de SO

A la digrafiya SO le corresponde, según la tabla TEN, el par ordenado $(20, 16)$. Ahora calculamos

$$f(20, 16) = \begin{pmatrix} 7 & 3 \\ 5 & 2 \end{pmatrix} \cdot \begin{pmatrix} 20 \\ 16 \end{pmatrix} \pmod{27} = \begin{pmatrix} 7 \cdot (20) + 3 \cdot (16) \\ 5 \cdot (20) + 2 \cdot (16) \end{pmatrix} \pmod{27} =$$

$$= \begin{pmatrix} 188 \\ 132 \end{pmatrix} \pmod{27} = \begin{pmatrix} 26 \\ 24 \end{pmatrix} \pmod{27}$$

Por lo tanto

$$f(20, 16) = (26, 24) \pmod{27}$$

Luego, al par ordenado $(26, 24)$ le corresponde, en la tabla TEN, a la digrafiya YW.

En consecuencia, la encriptación del mensaje VASO es BIYW.

El siguiente ejemplo ilustra cómo descryptar el mensaje:

Ejemplo 3.3

Desencriptar el mensaje *BIYW*, sabiendo que la matriz de digrafía es $A = \begin{pmatrix} 7 & 3 \\ 5 & 2 \end{pmatrix}$

Solución

La función a desencriptar es

$$g(x, y) = A^{-1} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \pmod{27}$$

Primeramente calculamos $A^{-1} \pmod{27}$. Entonces:

$$A^{-1} = \begin{pmatrix} 2 & -3 \\ -5 & 7 \end{pmatrix} \cdot (\det(A))^{-1} = \begin{pmatrix} 2 & -3 \\ -5 & 7 \end{pmatrix} \cdot (-1)^{-1} = \begin{pmatrix} -2 & 3 \\ 5 & -7 \end{pmatrix} \equiv \begin{pmatrix} 25 & 3 \\ 5 & 20 \end{pmatrix} \pmod{27}$$

Por lo tanto, la función a desencriptar es:

$$g(x, y) = \begin{pmatrix} 25 & 3 \\ 5 & 20 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \pmod{27}$$

La siguiente tabla muestra los datos obtenidos durante el proceso de desencriptado:

Digrafía del mensaje a descifrar	Código numérico de la digrafía	$g(x, y)$	Código alfabético de $g(x, y)$
BI	$(x, y) = (02, 09)$	$(23, 01)$	VA
YW	$(x, y) = (26, 24)$	$(20, 16)$	SO

Los detalles de su contenido los mostramos a continuación:

Primera Fila: Desencriptación de BI

A la digrafía *BI* le corresponde, según el código numérico en la tabla TEN, el par ordenado $(02, 09)$. A continuación calculamos

$$\begin{aligned} g(02, 09) &= \begin{pmatrix} 25 & 3 \\ 5 & 20 \end{pmatrix} \cdot \begin{pmatrix} 02 \\ 09 \end{pmatrix} \pmod{27} = \begin{pmatrix} 25 \cdot (02) + 3 \cdot (09) \\ 5 \cdot (02) + 20 \cdot (09) \end{pmatrix} \pmod{27} = \\ &= \begin{pmatrix} 77 \\ 190 \end{pmatrix} \pmod{27} = \begin{pmatrix} 23 \\ 01 \end{pmatrix} \pmod{27} \end{aligned}$$

Por lo tanto

$$g(02, 09) = (23, 01) \pmod{27}$$

Entonces al par ordenado $(23, 01)$ le corresponde, según la tabla TEN, a la digrafía VA.

Segunda Fila: Descriptación de YW

A la digrafía YW le corresponde, de acuerdo la tabla TEN, el par ordenado (26,24). Ahora calculamos

$$g(02,09) = \begin{pmatrix} 25 & 3 \\ 5 & 20 \end{pmatrix} \cdot \begin{pmatrix} 26 \\ 24 \end{pmatrix} \pmod{27} = \begin{pmatrix} 25 \cdot (26) + 3 \cdot (24) \\ 5 \cdot (26) + 20 \cdot (24) \end{pmatrix} \pmod{27} =$$
$$= \begin{pmatrix} 722 \\ 610 \end{pmatrix} \pmod{27} = \begin{pmatrix} 20 \\ 16 \end{pmatrix} \pmod{27}$$

Por lo tanto

$$g(02,09) = (20,16) \pmod{27}$$

Luego al par ordenado (20,16) le corresponde, en la tabla TEN, a la digrafía SO.

En consecuencia, la descriptación del mensaje BIYW es VASO.

Observación 3.4

Si el número de caracteres del texto a cifrar no fuera par, para encriptarlo se le agrega la última letra del mensaje. De este modo, el texto tendrá un número par de caracteres. Por ejemplo, si el mensaje a encriptar fuera SOL, se aplica el método de Matrices y Digrafas al texto SOLL.

Descriptado del Método de Matrices y Digrafas con Wolfram Mathematica 10.0

El siguiente programa, diseñado mediante el software Wolfram Mathematica 10.0, permite descifrar un mensaje encriptado con el Método de Matrices y Digrafas. El programa pide ingresar, en un cuadro de diálogo, los coeficientes de la matriz A junto con el mensaje a descifrar, según se muestra en la figura:

Ingrese los coeficientes de la matriz de digrafía:

3

7

2

5

Ingrese el mensaje a descifrar (entre comillas y con mayúsculas):

"MNOG"

Figura 3.4: Ingreso por pantalla de la matriz A y del mensaje cifrado.

Al ejecutar el programa, se comprueba que el mensaje descifrado es: *TODO*. El código fuente del programa lo mostramos a continuación.

***Código fuente del programa que descripta un mensaje cifrado con el
Método de Matrices y Digrafas***

```

Interpretation[{ Mensajec = "MNOG", A = MatrixForm[{{3, 7}, {2, 5}}],
aa = 3, bb = 7, cc = 2, dd = 5},
Panel [ Grid[{{Style [ "Ingrese los coeficientes de la matriz de digrafía:", Bold],
SpanFromLeft}, {" ", A = MatrixForm [{{InputField [ Dynamic[aa], FieldSize -> 2],
InputField [ Dynamic[bb], FieldSize -> 2]}, {InputField [
Dynamic [cc], FieldSize -> 2], InputField [ Dynamic[dd], FieldSize -> 2] }}} },
{ Style [ "Ingrese el mensaje a descriptar (entre comillas y con mayúsculas):", Bold],
SpanFromLeft }, {" ", InputField [Dynamic [ Mensajec] ] } } ] ],
Lista = {"A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L",
"M", "N", "Ñ", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z"};
Cadena = Characters [ Mensajec ]; n = Length [ Cadena ];
k = 0; While [ 27 k + (aa dd - bb cc) < 0, k ++ ];
deter = 27 k + (aa dd - bb cc);
If [ Mod [ deter, 27] == 0,
Print ["Lo siento, la matriz no es invertible en  $Z_{27}$  ],
Encontrado = 0;
i = 1;
While [ i <= n - 1 && Encontrado == 0,
If [ Mod [deter * i, n] == 1, Encontrado = 1]; i++;
];
invdeter = i - 1; Mensaje = "";
For [ i = 1, i <= n, i = i + 2,
Buscoletrau = Part [Cadena, i]; Buscoletrav = Part [Cadena, i + 1];
u = Part [ Part[ Position [Lista, Buscoletrau ], 1], 1];
v = Part [ Part[ Position [Lista, Buscoletrav ], 1], 1];
x = dd invdeter u - bb invdeter v;
y = - cc invdeter u + aa invdeter v;
k = 0; While [ 27 k + x < 0, k ++ ]; xx = 27 k + x; k = 0; While[27 k + x < 0, k++];
yy = 27 k + y;
If [ Mod [ xx, 27] == 0, Lugarx = 27, Lugarx = Mod [ xx, 27 ] ];
Letrax = Part [ Lista, Lugarx ];
Mensaje = StringInsert [ Mensaje, Letrax, Length [ Characters[Mensaje] ] + 1];
If [ Mod [ yy, 27] == 0, Lugary = 27, Lugary = Mod [ yy, 27] ];
Letray = Part [ Lista, Lugary ];
Mensaje = StringInsert [ Mensaje, Letray, Length [ Characters [Mensaje] ] + 1];
];
Print [ "El texto descriptado es:" ]; Print [ Mensaje]
]
]

```

Ventaja del Método de Matrices y Digrafías con respecto a los Métodos de Julio César y de Julio César Mejorado

En esta sección analizaremos la seguridad del Método de Matrices y Digrafías en relación a los métodos de Julio César y de Julio César Mejorado. La seguridad de un método radica en la dificultad para descubrir la clave. Mientras mayor sea la cantidad de claves que brinde un método, más difícil será encontrar el mensaje descifrado. Ello se debe a que hay que probar con todas las posibles claves hasta hallar un “texto coherente” que indique que ese será el mensaje descifrado. De este modo, si el número de claves fuera muy grande, el tiempo que se tardaría en descifrar un mensaje podría ser inmanejable. La siguiente tabla, cuyos datos fueron extraídos de [4] (Trabajo de Tesis de Cecilia Prieto), muestra la cantidad de claves que provee el método de Julio César y el de Julio César Mejorado.

<i>Método Criptográfico</i>	<i>Función encriptadora</i>	<i>Clave</i>	<i>Cantidad de claves que provee el método</i>
<i>Julio César</i>	$f(x) = x + k$ Con $k \in \mathbb{Z}_{27} \wedge k \neq 0$	k	26
<i>Julio César Mejorado</i>	$f(x) = a.x + b$ Con $a, b \in \mathbb{Z}_{27}$ a invertible en $\mathbb{Z}_{27}$	(a, b)	485

Como podemos comprobar, el Método de Julio César Mejorado es más seguro que el de Julio César pero, aún así, con el uso de ordenadores es muy fácil de romper.

Analizaremos ahora la cantidad de claves que dispone el Método de Matrices y Digrafías. En este caso, la clave es la matriz

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

Con $a, b, c, d \in \mathbb{Z}_{27} \wedge \det(A)$ invertible en $\mathbb{Z}_{27}$

El siguiente programa, realizado con Wolfram Mathematica 10.0, cuenta cuántas matrices de este tipo hay:

```
Cont = 0 ;
For [ a = 0, a <= 26, a ++ ,
  For [ b = 0, b <= 26, b ++ ,
    For [ c = 0, c <= 26, c ++ ,
      For [ d = 0, d <= 26, d ++ ,
        If [ GCD [ a d - b c, 27 ] == 1, Cont = Cont + 1 ] ] ] ]
Print [ "El número de matrices de orden 2 invertibles en  $\mathbb{Z}_{27}$  es:", Cont]
```

Al ejecutar el programa, vemos que existen 314928 matrices de este tipo. Es decir, el Método de Matrices y Digrafas provee 314928 claves distintas.

Claramente, este método es muchísimo más seguro que el de Julio César y el de Julio César Mejorado. Sin embargo, con el uso de computadoras, el mensaje cifrado también puede descubrirse en un tiempo prudencial.

En general, con el advenimiento de la Era Informática, los ordenadores han hecho vulnerables a la mayoría de los algoritmos criptográficos simétricos. Es por esta razón que fueron dejados de lado para dar lugar a la Criptografía Asimétrica, siendo el Algoritmo de Deffie Hellman uno de sus principales representantes.

En el próximo capítulo expondremos detalladamente este algoritmo, ya que en él se basa el algoritmo que desarrollaremos en este trabajo de tesis.

Capítulo 4

Algoritmo de Diffie-Hellman

Capítulo 4: Algoritmo de Diffie-Hellman

En este capítulo analizaremos detalladamente el Algoritmo de Diffie-Hellman, desarrollado en 1976 por Whitfield Diffie y Martin Hellman. Es utilizado para generar una clave secreta, que será compartida por dos personas. No se lo puede emplear para encriptar mensajes, conversaciones o firmas digitales.



Figura 4.1: Los autores del Algoritmo de Diffie-Hellman

Fuente: <http://tecnoswaldo.blogspot.com/2015/10/whitfield-diffie-y-martin-hellman.html>

La principal aplicación de este protocolo es acordar una clave con la que posteriormente se puedan establecer comunicaciones secretas entre dos máquinas.

Su seguridad radica en la extrema dificultad de calcular ciertos logaritmos discretos los cuales, hasta el día de hoy, constituyen un problema intratable para la Computación Clásica.

Este tipo de algoritmo pertenece a la rama de la Criptografía Asimétrica (o Criptografía de llave pública). Recordemos que ésta es una Criptografía que se basa en emplear dos claves: una clave pública, conocida por todo el mundo y una clave privada, conocida por una única persona. En el caso del Algoritmo de Diffie-Hellman, ambas claves se combinan para generar una clave secreta.

La Criptografía Asimétrica ofrece mayores niveles de seguridad en comparación con la Simétrica. Su fiabilidad se debe a la dificultad para resolver ciertos problemas matemáticos, algo que no ocurre con la Criptografía Simétrica. Además, proporciona confidencialidad, autenticación e integridad.

Procedimiento del algoritmo

Como mencionamos anteriormente, este algoritmo permite generar una clave secreta para ser compartida por dos personas, a las que llamaremos **Alicia** y **Bernardo**. Consta de los siguientes pasos, diagramados en la siguiente figura:

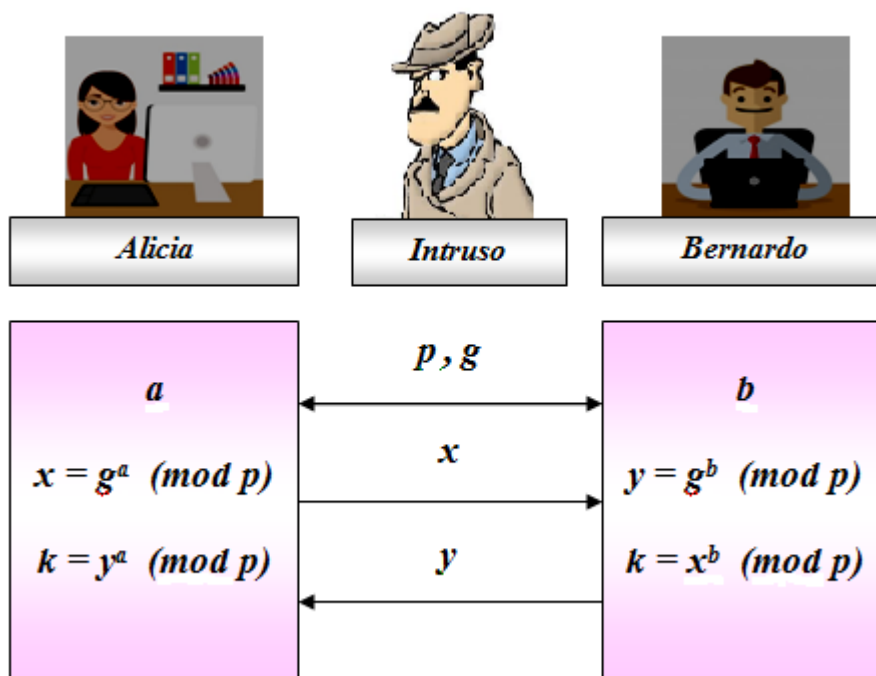


Figura 4.2: Diagrama que ilustra los pasos del Algoritmo de Diffie-Hellman.

Fuente de las imágenes humanas: <http://deadislive-tk.blogspot.com/2010/>

Primer Paso

A través de un medio público (Internet por ejemplo) Alicia y Bernardo se ponen de acuerdo en usar un número primo p muy grande y un número g menor que p . El “intruso” puede detectar ambos números.

Segundo Paso

Alicia escoge un número natural “ a ” menor que p que usará como exponente y lo mantiene en secreto. Bernardo hace lo propio, escoge un número natural “ b ” menor que p que usará como exponente y lo mantiene en secreto.

Tercer Paso

Con su exponente secreto Alicia calcula $x = g^a \pmod{p}$ y envía el valor “ x ” a Bernardo. De este modo, Bernardo conoce “ x ” pero no conoce “ a ”. Bernardo por su parte calcula $y = g^b \pmod{p}$ y envía el valor “ y ” a Alicia. De este modo, el intruso puede ver los resultados “ x ” e “ y ” pero no ve “ a ” ni “ b ”.

Cuarto Paso

Alicia recibe “y” y con su exponente secreto calcula $k = y^a \pmod{p}$.

Bernardo hace lo propio: recibe “x” y con su exponente secreto calcula $k = x^b \pmod{p}$. De este modo, Alicia y Bernardo comparten en forma secreta la misma clave **k**.

Justificación de porqué Alicia y Bernardo obtienen la misma clave “k”.

Como

$$y^a = (g^b)^a = g^{b \cdot a} \pmod{p}$$

Y por otro lado

$$x^b = (g^a)^b = g^{a \cdot b} = g^{b \cdot a} \pmod{p}$$

Resulta entonces que $y^a = x^b = k \pmod{p}$.

Seguridad del Algoritmo de Deffie-Hellman

El intruso conoce **p**, **g**, **x**, **y**. No conoce **a** ni **b**. Conociendo “a” podría descubrir la clave **k** ya que $y^a = k \pmod{p}$.

Como $g^a = x \pmod{p}$ entonces $a = \log_g(x) \pmod{11}$. De este modo, para hallar “a”, el intruso debería calcular un logaritmo discreto. Para ello podría recurrir a la computadora y calcular las sucesivas potencias de $g : g^1, g^2, g^3, g^4, \dots$ hasta que le dé **x**. El Teorema de Euler asegura que dicho exponente es menor que $p-1$. Pero si el primo **p** tiene más de 200 dígitos, el proceso puede tomar años.

Por esta razón, si el número primo **p** es suficientemente grande, el Algoritmo de Deffie-Hellman es muy seguro debido a que es prácticamente imposible descifrar su clave en un tiempo prudencial.

Una desventaja del Algoritmo de Deffie-Hellman

El problema fundamental de este algoritmo es que es sensible a ataques activos del tipo “hombre en el medio”.

El “hombre en el medio” o “intruso” podría interceptar la comunicación entre Alicia y Bernardo. Así, podría hacerse pasar por Alicia frente a Bernardo y por Bernardo frente a Alicia. Este riesgo es factible porque el algoritmo no dispone de ningún mecanismo para validar la identidad de los participantes en la comunicación.

De este modo, el atacante puede acordar una clave con Alicia y otra con Bernardo.

Una vez establecidas las dos claves, el intruso haría de puente entre Alicia y Bernardo, escuchando la conversación en ambos sentidos y retransmitiendo los datos entre ellos.

Para corregir la vulnerabilidad del protocolo, éste debe ser utilizado conjuntamente con algún sistema que autentique los mensajes.

En el siguiente Ejemplo mostraremos cómo implementar el protocolo Deffie-Hellman. El cálculo de las potencias modulares se llevará a cabo aplicando el algoritmo descrito en el Capítulo 2.

Ejemplo 4.1

Alicia y Bernardo acuerdan usar el primo $p=2131$ y $g=102$. Alicia escoge su exponente $a=381$, no se lo revela a nadie y calcula $x = g^a = (102)^{381} = 737 \pmod{2131}$ y envía “737” a Bernardo. Por otro lado, Bernardo escoge su exponente $b=751$, no se lo revela a nadie, calcula $y = g^b = (102)^{751} = 588 \pmod{2131}$ y envía “588” a Alicia. Alicia, en secreto, hace el cálculo $(588)^{381} = 1175 \pmod{2131}$. Bernardo, en secreto, hace el cálculo $(737)^{751} = 1175 \pmod{2131}$. De este modo, $k = 1175$ es la clave que comparten. El intruso conoce $p = 2131$, $g = 102$, $x = 737$, $y = 588$. Su problema es hallar $a = \log_g(x) \pmod{p}$ ó $b = \log_g(y) \pmod{p}$. Con un número p de 4 cifras, el algoritmo que calcula las potencias modulares expuesto en el Capítulo 2 puede dar la respuesta. Pero si p tuviera 100 cifras o más, el cálculo de cualquiera de los dos logaritmos sería muy costoso en términos de tiempo.

El próximo Ejemplo mostraremos cómo el “intruso” podría llegar a descubrir la clave secreta si el número p fuera pequeño.

Ejemplo 4.2

Un intruso navega por los portales de Alicia y Bernardo y en ambos halla $p = 13$ y $g = 7$. Luego, ve el misterioso número 3 en la página de Bernardo y el misterioso número 2 en la de Alicia. ¿Podría el intruso determinar el número secreto que Alicia y Bernardo comparten?

Solución

Se sabe que $p = 13$ y $g = 7$. También que $x = 2$ e $y = 3$.

Entonces, se procede a calcular **a** y **b**. Como

$$a = \log_g(x) \pmod{p} \Rightarrow a = \log_7(2) \pmod{13}$$

Por lo tanto, mediante el algoritmo del Capítulo 2, se calcularán las potencias modulares $7^i \pmod{13}$ hasta encontrar un “i” que verifique que $7^i = 2 \pmod{13}$. La siguiente tabla muestra los resultados correspondientes:

i	0	1	2	3	4	5	6	7	8	9	10	11
$7^i \pmod{13}$	1	7	10	5	9	11	12	6	3	8	4	2

Luego, $a = \log_7 (2) = 11 \pmod{13}$

Por otro lado,

$$b = \log_7 (3) \pmod{13} \Rightarrow b = \log_7 (3) \pmod{13}$$

Observando la tabla anterior, se ve que $b = \log_7 (3) = 8 \pmod{13}$

Finalmente, se calcula la clave k que Alicia y Bernardo comparten:

$$k = x^b = 2^8 = 9 \pmod{13}$$

$$k = y^a = 3^{11} = 9 \pmod{13}$$

Por lo tanto, el número secreto que Alicia y Bernardo comparten es $k = 9$.

Capítulo 5

Algoritmo de Diffie-Hellman empleando anillos no conmutativos

Capítulo 5: El Algoritmo de Diffie-Hellman aplicado a anillos no conmutativos

Introducción

La Criptografía Asimétrica (AC) ha estado presente desde los trabajos pioneros de Diffie, Pohlig, Merkel y Hellman.

Prácticamente, todos los algoritmos basados en AC están vinculados con el problema de factorización de números enteros o con el cálculo de logaritmos discretos. Ambos problemas se relacionan con operaciones aritméticas definidas en estructuras algebraicas conmutativas (grupos o anillos conmutativos). Una de las desventajas de trabajar con este tipo de estructuras es que se conocen ataques de complejidad subexponencial y ataques eficientes a través de algoritmos cuánticos.

Por esta razón hay serios motivos para desconfiar de los mismos. Para reemplazar de raíz a la AC convencional ha surgido, en estos últimos años, un interés creciente en aplicar estructuras algebraicas no conmutativas, conocida como “Criptografía Pos Cuántica”. Con este objetivo, se comenzaron a utilizar los anillos de polinomios matriciales, que es lo que aplicaremos en este trabajo.

Desventajas de los algoritmos criptográficos basados en estructuras algebraicas no conmutativas

Existen inconvenientes que subsisten en la Criptografía Asimétrica que emplea Estructuras Algebraicas No Conmutativas (NCAS). Uno de ellos es la necesidad de emplear bibliotecas de precisión extendida y el uso de un hardware con alta capacidad computacional. Un desafío abierto es hallar un modelo de NCAS que sea criptográficamente seguro pero capaz de emplear recursos computacionales escasos, lo que permitirá una eficaz operación en plataformas de baja potencia. En virtud de estas observaciones, surge el siguiente concepto:

Definición 5.1: Modelo criptográfico compacto

Un modelo criptográfico compacto es un algoritmo criptográfico asimétrico que emplea estructuras algebraicas no conmutativas (NCAS) y recursos computacionales escasos (procesadores y memoria de poca cantidad de bits).

Modelo Criptográfico Compacto propuesto en este trabajo

En el modelo compacto que propondremos, utilizaremos como estructura algebraica no conmutativa al anillo de matrices de orden 4 a coeficientes en $\mathbb{Z}_{256}$. El algoritmo criptográfico asimétrico que plantearemos será el de Deffie-Hellman, pero adaptado a este tipo de estructuras.

Para implementar este algoritmo sólo se precisarán procesadores de 16 bits en controladores inteligentes como smartcards, teléfonos celulares o llaves inteligentes USB.

Antes de describir el algoritmo, desarrollaremos los siguientes conceptos matemáticos que aplicaremos para justificar el modelo.

Nociones matemáticas previas

Definición 5.2: El anillo de matrices a coeficientes en $\mathbb{Z}_{256}$

Sea el conjunto de matrices $M_4[\mathbb{Z}_{256}]$, es decir,

$$M_4[\mathbb{Z}_{256}] = \left\{ A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \text{ tal que } a_{ij} \in \mathbb{Z}_{256} \quad \forall i, j=1,2,3,4 \right\}$$

Entonces la terna $(M_4[\mathbb{Z}_{256}], +, \cdot)$ es un anillo no conmutativo donde $+$ y $\cdot$ son, respectivamente, la suma y producto matricial, pero con las operaciones aritméticas calculadas en $\mathbb{Z}_{256}$.

Ejemplo 5.3

Sean $A, B \in M_4[\mathbb{Z}_{256}]$ definidas por

$$A = \begin{pmatrix} 15 & 240 & 8 & 11 \\ 120 & 7 & 2 & 94 \\ 135 & 75 & 4 & 3 \\ 25 & 5 & 10 & 9 \end{pmatrix}, \quad B = \begin{pmatrix} 5 & 210 & 6 & 15 \\ 12 & 72 & 3 & 4 \\ 130 & 66 & 2 & 5 \\ 33 & 15 & 11 & 8 \end{pmatrix}$$

Calcular $A \cdot B \pmod{256}$

Solución

$$A \cdot B \pmod{256} = C = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{pmatrix}$$

$$\text{Donde } c_{ij} = \sum_{k=1}^4 a_{ik} \cdot b_{kj} \pmod{256}, \forall i, j=1,2,3,4$$

Entonces

$$A \cdot B = \begin{pmatrix} 15 & 240 & 8 & 11 \\ 120 & 7 & 2 & 94 \\ 135 & 75 & 4 & 3 \\ 25 & 5 & 10 & 9 \end{pmatrix} \cdot \begin{pmatrix} 5 & 210 & 6 & 15 \\ 12 & 72 & 3 & 4 \\ 130 & 66 & 2 & 5 \\ 33 & 15 & 11 & 8 \end{pmatrix} = \begin{pmatrix} 4328 & 21123 & 947 & 1313 \\ 3806 & 27246 & 1779 & 2590 \\ 1924 & 34059 & 1076 & 2369 \\ 1732 & 6405 & 284 & 517 \end{pmatrix} \equiv$$

$$\equiv \begin{pmatrix} 232 & 131 & 179 & 33 \\ 222 & 110 & 243 & 30 \\ 132 & 11 & 52 & 65 \\ 196 & 5 & 28 & 5 \end{pmatrix} \pmod{256}$$

Por lo tanto

$$A \cdot B = \begin{pmatrix} 232 & 131 & 179 & 33 \\ 222 & 110 & 243 & 30 \\ 132 & 11 & 52 & 65 \\ 196 & 5 & 28 & 5 \end{pmatrix} \pmod{256}$$

En la siguiente Proposición mostraremos cuántos elementos tiene el conjunto $M_4[\mathbb{Z}_{256}]$ y qué cantidad de memoria ocupa cada matriz de dicho conjunto.

Proposición 5.4

Dado el conjunto de matrices $M_4[\mathbb{Z}_{256}]$ entonces

1) Su cardinal es 2^{128} .

2) Cada matriz de $M_4[\mathbb{Z}_{256}]$ ocupa 16 bytes de memoria.

Demostración

$$\text{Sea } A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \in M_4 [\mathbb{Z}_{256}]. \text{ Entonces}$$

1) Como $a_{ij} \in \mathbb{Z}_{256} = \{0, 1, \dots, 255\} \forall i, j \in \{1, 2, 3, 4\}$, a_{ij} puede tomar $256 = 2^8$ valores distintos. Debido a que la matriz A contiene 16 coeficientes entonces, por el Principio de la Multiplicación, el total de los posibles valores que puede tomar la matriz A es $\underbrace{2^8 \cdot 2^8 \cdot \dots \cdot 2^8}_{16 \text{ veces}} = (2^8)^{16} = 2^{128}$. Luego, $M_4 [\mathbb{Z}_{256}]$ posee 2^{128} elementos.

2) Nuevamente, como $a_{ij} \in \mathbb{Z}_{256} \forall i, j \in \{1, 2, 3, 4\}$, el mayor valor que se le puede asignar a a_{ij} es 255. Por lo demostrado en el Ejercicio 2.24, el número 255 ocupa 1 byte de memoria. En consecuencia, debido a que la unidad mínima de almacenamiento de datos es el byte, cada a_{ij} abarca 1 byte.

Como la matriz A contiene 16 elementos, resulta que se necesitan 16 bytes de memoria para almacenar A . ■

A continuación definiremos algunas operaciones con los elementos de un Anillo, que serán utilizadas en el algoritmo criptográfico que desarrollaremos.

Definición 5.5: Producto de un entero por un elemento de un Anillo

Sean $(A, +, \cdot)$ un anillo, $k \in \mathbb{Z}$ y $a \in A$.

1) Si $k > 0$ entonces

$$(k)a = \underbrace{a + a + \dots + a}_{k \text{ veces}}$$

2) Si $k < 0$ entonces

$$(k)a = (-k)(-a) = \underbrace{(-a) + (-a) + \dots + (-a)}_{-k \text{ veces}}$$

3) Si $k = 0$ entonces

$$(k)a = 0$$

Donde 0 es el elemento neutro para la operación +.

Definición 5.6: Potencia de un elemento de un Anillo

Sean $(A, +, \cdot)$ un anillo con identidad, $k \in \mathbb{Z}$ y $a \in A$.

1) Si $k > 0$ entonces

$$a^k = \underbrace{a \cdot a \cdot \dots \cdot a}_{k \text{ veces}}$$

2) Si $k < 0$ y a es invertible entonces

$$a^k = (a^{-1})^{(-k)} = \underbrace{(a^{-1}) \cdot (a^{-1}) \cdot \dots \cdot (a^{-1})}_{-k \text{ veces}}$$

3) Si $k = 0$ entonces

$$a^k = 1$$

Donde 1 es el elemento neutro para la operación $\cdot$.

Observación 5.7

De acuerdo a la Definición 5.5, si $a \in A$ y $k > 0$, el producto de un entero positivo por un elemento de un anillo puede escribirse como:

$$(k)a = \sum_{i=1}^k a$$

Basándonos en esta expresión, probaremos la siguiente Proposición

Proposición 5.8

Sean $(A, +, \cdot)$ un anillo, $m, n \in \mathbb{N} \wedge a \in A$. Entonces

$$\sum_{i=1}^m \sum_{j=1}^n a = (m \cdot n)a$$

Demostración

$$\sum_{i=1}^m \sum_{j=1}^n a = \sum_{i=1}^m (n)a = \underbrace{(n)a + (n)a + \dots + (n)a}_{m \text{ veces}} =$$

$$= \underbrace{a + a + \dots + a}_{n \text{ veces}} + \underbrace{a + a + \dots + a}_{n \text{ veces}} + \dots + \underbrace{a + a + \dots + a}_{n \text{ veces}} = \underbrace{a + a + \dots + a}_{m \cdot n \text{ veces}} = (m \cdot n)a$$

Por lo tanto $\sum_{i=1}^m \sum_{j=1}^n a = (m \cdot n)a$

■

Proposición 5.9

Sean $(A, +, \cdot)$ un anillo con identidad, $k_1, k_2 \in \mathbb{N}$, $m, n \in \mathbb{Z}_{\geq 0} \wedge a \in A$. Entonces

$$1) (k_1)a^m \cdot (k_2)a^n = (k_1 k_2)a^{m+n}$$

$$2) (k_1)a^m \cdot (k_2)a^n = (k_2)a^n \cdot (k_1)a^m$$

Demostración

1) Aplicando la Observación 5.7, la propiedad distributiva de la suma con respecto al producto de elementos de un anillo y la Proposición 5.8, resulta

$$(k_1)a^m \cdot (k_2)a^n = \left(\sum_{i=1}^{k_1} a^m \right) \cdot \left(\sum_{j=1}^{k_2} a^n \right) = \sum_{i=1}^{k_1} \sum_{j=1}^{k_2} a^m \cdot a^n = \sum_{i=1}^{k_1} \sum_{j=1}^{k_2} a^{m+n} = (k_1 k_2)a^{m+n}$$

2) Del inciso anterior obtuvimos

$$(k_1)a^m \cdot (k_2)a^n = \sum_{i=1}^{k_1} \sum_{j=1}^{k_2} a^{m+n} \quad [1]$$

Por otro lado, por la propiedad conmutativa de la suma de elementos de un anillo, se tiene

$$\sum_{i=1}^{k_1} \sum_{j=1}^{k_2} a^{m+n} = \sum_{j=1}^{k_2} \sum_{i=1}^{k_1} a^{m+n} \quad [2]$$

Reemplazando [2] en [1], utilizando la Proposición 5.8 y el inciso 1) obtenemos

$$(k_1)a^m \cdot (k_2)a^n = \sum_{j=1}^{k_2} \sum_{i=1}^{k_1} a^{m+n} = (k_2 k_1)a^{m+n} = (k_2)a^n \cdot (k_1)a^m \quad \blacksquare$$

Observación 5.10

Si el anillo $(A, +, \cdot)$ no es conmutativo, en general no se verifica que

$$(k_1)a \cdot (k_2)b = (k_2)b \cdot (k_1)a$$

$\forall k_1, k_2 \in \mathbb{N} \wedge a, b \in A$ con $a \neq b$.

Por ejemplo, si consideramos $(M_2(\mathbb{R}), +, \cdot)$, el anillo no conmutativo de matrices de orden 2 a coeficientes reales, y tomamos $k_1 = 2, k_2 = 3$, $A = \begin{pmatrix} 1 & 2 \\ 3 & 5 \end{pmatrix}$, $B = \begin{pmatrix} 2 & 1 \\ -1 & 1 \end{pmatrix}$ entonces se puede verificar que

$$(k_1)A \cdot (k_2)B \neq (k_2)B \cdot (k_1)A$$

Pues

$$(2) \begin{pmatrix} 1 & 2 \\ 3 & 5 \end{pmatrix} \cdot (3) \begin{pmatrix} 2 & 1 \\ -1 & 1 \end{pmatrix} \neq (3) \begin{pmatrix} 2 & 1 \\ -1 & 1 \end{pmatrix} \cdot (2) \begin{pmatrix} 1 & 2 \\ 3 & 5 \end{pmatrix}$$

La no conmutatividad de un anillo es crucial para el modelo criptográfico que desarrollaremos.

Definición 5.11: Polinomios definidos en un Anillo

Sea $(A, +, \cdot)$ un anillo con identidad. Un polinomio en la variable x , a coeficientes enteros no negativos y de grado menor o igual que n , es una expresión de la forma

$$f(x) = \sum_{i=0}^n (a_i) x^i = (a_0)1 + (a_1)x^1 + \dots + (a_n)x^n$$

Donde

$$x \in A$$

$$n, a_0, a_1, \dots, a_n \in \mathbb{Z}_{\geq 0}$$

Al conjunto formado por todos estos polinomios lo llamaremos $\mathbb{Z}_{\geq 0}[A]$. Su definición es la siguiente:

Definición 5.12: El conjunto $\mathbb{Z}_{\geq 0}[A]$

Sea $(A, +, \cdot)$ un anillo. Entonces

$$\mathbb{Z}_{\geq 0}[A] = \left\{ f(x) = \sum_{i=0}^n (a_i) x^i \text{ tal que } x \in A \wedge n, a_0, a_1, \dots, a_n \in \mathbb{Z}_{\geq 0} \right\}$$

La siguiente Proposición es también relevante para fundamentar el modelo que desarrollaremos.

Proposición 5.13

Sean $(A, +, \cdot)$ un anillo con identidad, $f(x), h(x) \in \mathbb{Z}_{\geq 0}[A] \wedge a \in A$.

Entonces

$$f(a) \cdot h(a) = h(a) \cdot f(a)$$

Demostración

Como $f(x), h(x) \in \mathbb{Z}_{\geq 0}[A]$ entonces $f(x) = \sum_{i=0}^n (a_i) x^i \wedge h(x) = \sum_{j=0}^m (b_j) x^j$

Por lo tanto, aplicando la propiedad distributiva del producto con respecto a la suma de elementos de un anillo, la Proposición 5.9 inciso 2) y la propiedad conmutativa de la suma, se tiene

$$\begin{aligned} f(a) \cdot h(a) &= \left(\sum_{i=0}^n (a_i) a^i \right) \cdot \left(\sum_{j=0}^m (b_j) a^j \right) = \sum_{i=0}^n \sum_{j=0}^m ((a_i) a^i) \cdot ((b_j) a^j) = \\ &= \sum_{i=0}^n \sum_{j=0}^m ((b_j) a^j) \cdot ((a_i) a^i) = \sum_{j=0}^m \sum_{i=0}^n ((b_j) a^j) \cdot ((a_i) a^i) = \left(\sum_{j=0}^m (b_j) a^j \right) \cdot \left(\sum_{i=0}^n (a_i) a^i \right) = \\ &= h(a) \cdot f(a). \end{aligned}$$

■

Observación 5.14

Si el anillo $(A, +, \cdot)$ no es conmutativo, en general no se verifica que

$$f(a) \cdot h(b) = h(b) \cdot f(a)$$

Si $a, b \in A$ con $a \neq b$.

Este detalle es también importante para que el algoritmo sea seguro.

Protocolo de intercambio de claves Diffie-Hellman usando anillos no conmutativos

En esta sección presentaremos una variación del algoritmo de Diffie-Hellman aplicado al anillo no conmutativo $(M_4[\mathbb{Z}_{256}], +, \cdot)$. Como veremos, esto permitirá mejorar el tiempo requerido para generar claves, en relación al algoritmo de Diffie-Hellman clásico. Además, podrá ser ejecutado en dispositivos de baja capacidad computacional.

Procedimiento del algoritmo

Alicia y Bernardo generarán una clave de seguridad para ser compartida. Los pasos a seguir los mostramos en el siguiente gráfico:

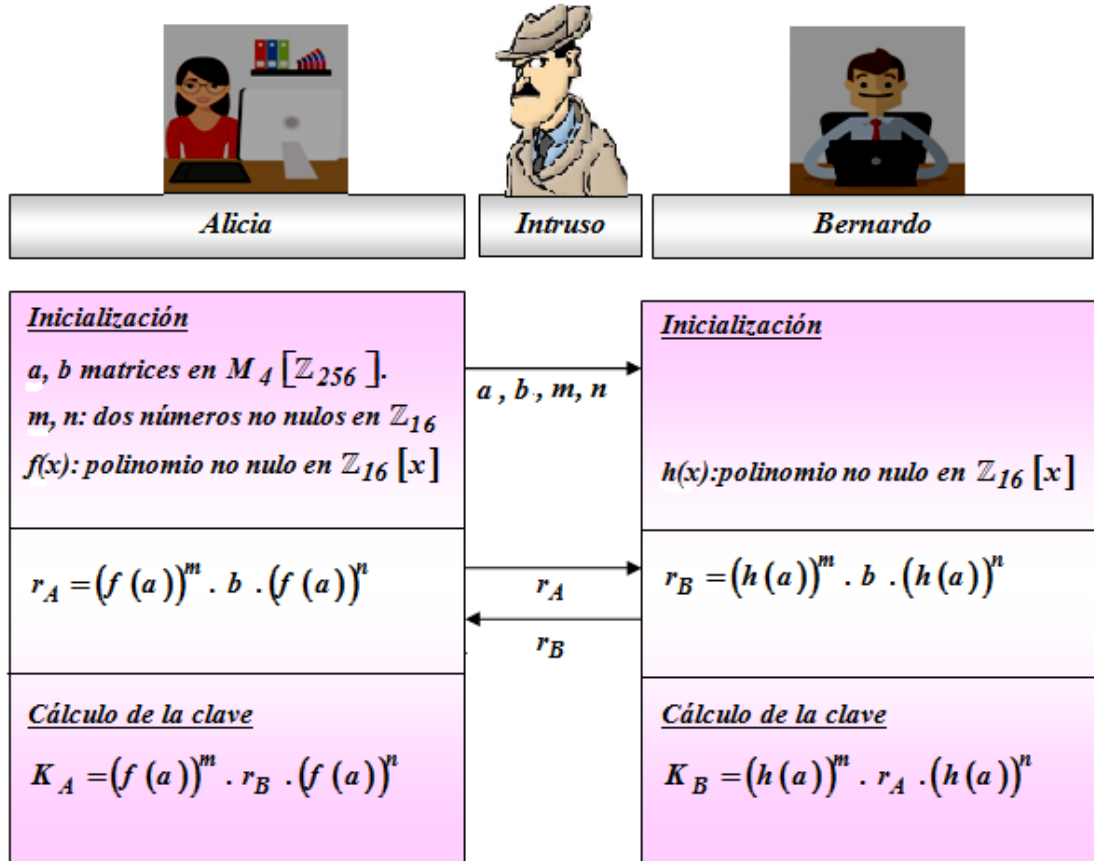


Figura 5.1: Diagrama que ilustra los pasos del Algoritmo de Diffie-Hellman aplicando matrices.

Primer Paso

Alicia elige dos matrices a y b no nulas, tales que $a, b \in M_4[\mathbb{Z}_{256}]$. Además, elige dos números no nulos m y n pertenecientes a $\mathbb{Z}_{16}$. Luego envía a Bernardo a, b, m , y n .

Segundo Paso

Alicia elige al azar un polinomio no nulo $f(x) \in \mathbb{Z}_{16}[x]$ tal que $f(a) \neq \bar{0}$ (donde $\bar{0}$ es la matriz nula de orden 4) y sus coeficientes y exponentes pertenezcan a $\mathbb{Z}_{16}$. En este caso, $f(x)$ será su clave privada.

Tercer Paso

Bernardo elige al azar un polinomio no nulo $h(x) \in \mathbb{Z}_{16}[x]$ tal que $h(a) \neq \bar{0}$ y sus coeficientes y exponentes pertenezcan a $\mathbb{Z}_{16}$. En este caso, $h(x)$ será su clave privada.

Cuarto Paso

Alicia calcula

$$r_A = (f(a))^m \cdot b \cdot (f(a))^n$$

Y se lo envía por un canal inseguro a Bernardo.

Quinto Paso

Bernardo computa

$$r_B = (h(a))^m \cdot b \cdot (h(a))^n$$

Y se lo envía por igual canal a Alicia.

Sexto Paso

Alicia calcula su clave compartida

$$K_A = (f(a))^m \cdot r_B \cdot (f(a))^n$$

Séptimo Paso

Bernardo calcula su clave compartida

$$K_B = (h(a))^m \cdot r_A \cdot (h(a))^n$$

Justificación de porqué Alicia y Bernardo obtienen la misma clave .

Observemos que $K_A = K_B$ pues, por la Proposición 5.13 y la propiedad asociativa del producto entre elementos del anillo $(M_4[\mathbb{Z}_{256}], +, \cdot)$, se verifica

$$\begin{aligned} K_A &= (f(a))^m \cdot r_B \cdot (f(a))^n = (f(a))^m \cdot (h(a))^m \cdot b \cdot (h(a))^n \cdot (f(a))^n = \\ &= (h(a))^m \cdot (f(a))^m \cdot b \cdot (f(a))^n \cdot (h(a))^n = (h(a))^m \cdot r_A \cdot (h(a))^n = K_B \end{aligned}$$

En el siguiente Ejemplo mostraremos en detalle cómo generar una clave aplicando este algoritmo. Recordemos que todas las operaciones numéricas se efectúan en $\mathbb{Z}_{256}$.

Un ejemplo del protocolo Deffie-Hellman con matrices

Alicia y Bernardo se pusieron de acuerdo para generar una clave secreta que compartirán. Para ello han decidido aplicar el protocolo de Deffie-Hellman con matrices. Los pasos a seguir son los siguientes:

Primer Paso

Alicia elige dos matrices $a, b \in M_4 [\mathbb{Z}_{256}]$ no nulas, donde

$$a = \begin{pmatrix} 17 & 3 & 128 & 45 \\ 255 & 12 & 1 & 65 \\ 136 & 90 & 28 & 13 \\ 201 & 166 & 8 & 28 \end{pmatrix}, b = \begin{pmatrix} 244 & 25 & 67 & 117 \\ 9 & 25 & 0 & 59 \\ 11 & 2 & 195 & 219 \\ 104 & 38 & 121 & 44 \end{pmatrix}.$$

Además, elige dos números no nulos $m=12$ y $n=7$ pertenecientes a $\mathbb{Z}_{16}$. Luego envía a Bernardo a, b, m , y n .

Segundo Paso

Alicia elige su clave privada $f(x) = 11x^{14} + 6x^9 + 3x^7 + 13$.

Tercer Paso

Bernardo elige su clave privada $h(x) = x^{15} + 15x^{12} + 3x^8 + 11x^5 + 2x + 7$.

Cuarto Paso

Alicia calcula su token

$$r_A = (f(a))^m \cdot b \cdot (f(a))^n$$

Y se lo envía por un canal inseguro a Bernardo. En este caso,

$$f(a) = 11 \begin{pmatrix} 17 & 3 & 128 & 45 \\ 255 & 12 & 1 & 65 \\ 136 & 90 & 28 & 13 \\ 201 & 166 & 8 & 28 \end{pmatrix}^{14} + 6 \begin{pmatrix} 17 & 3 & 128 & 45 \\ 255 & 12 & 1 & 65 \\ 136 & 90 & 28 & 13 \\ 201 & 166 & 8 & 28 \end{pmatrix}^9 + 3 \begin{pmatrix} 17 & 3 & 128 & 45 \\ 255 & 12 & 1 & 65 \\ 136 & 90 & 28 & 13 \\ 201 & 166 & 8 & 28 \end{pmatrix}^7 + 13.I = \begin{pmatrix} 101 & 70 & 91 & 29 \\ 163 & 208 & 97 & 193 \\ 111 & 46 & 127 & 167 \\ 230 & 1 & 68 & 106 \end{pmatrix}$$

Donde I es la matriz identidad de orden 4.

$$r_A = \begin{pmatrix} 101 & 70 & 91 & 29 \\ 163 & 208 & 97 & 193 \\ 111 & 46 & 127 & 167 \\ 230 & 1 & 68 & 106 \end{pmatrix}^{12} \begin{pmatrix} 244 & 25 & 67 & 117 \\ 9 & 25 & 0 & 59 \\ 11 & 2 & 195 & 219 \\ 104 & 38 & 121 & 44 \end{pmatrix} \begin{pmatrix} 101 & 70 & 91 & 29 \\ 163 & 208 & 97 & 193 \\ 111 & 46 & 127 & 167 \\ 230 & 1 & 68 & 106 \end{pmatrix}^7 =$$

$$= \begin{pmatrix} 214 & 107 & 138 & 118 \\ 4 & 243 & 148 & 80 \\ 12 & 233 & 60 & 176 \\ 73 & 202 & 107 & 91 \end{pmatrix}$$

Quinto Paso

Bernardo computa su token

$$r_B = (h(a))^m \cdot b \cdot (h(a))^n$$

Y se lo envía por igual canal a Alicia. En este caso,

$$h(a) = \begin{pmatrix} 17 & 3 & 128 & 45 \\ 255 & 12 & 1 & 65 \\ 136 & 90 & 28 & 13 \\ 201 & 166 & 8 & 28 \end{pmatrix}^{15} + 15 \begin{pmatrix} 17 & 3 & 128 & 45 \\ 255 & 12 & 1 & 65 \\ 136 & 90 & 28 & 13 \\ 201 & 166 & 8 & 28 \end{pmatrix}^{12} + 3 \begin{pmatrix} 17 & 3 & 128 & 45 \\ 255 & 12 & 1 & 65 \\ 136 & 90 & 28 & 13 \\ 201 & 166 & 8 & 28 \end{pmatrix}^8 +$$

$$+ 11 \begin{pmatrix} 17 & 3 & 128 & 45 \\ 255 & 12 & 1 & 65 \\ 136 & 90 & 28 & 13 \\ 201 & 166 & 8 & 28 \end{pmatrix}^5 + 2 \begin{pmatrix} 17 & 3 & 128 & 45 \\ 255 & 12 & 1 & 65 \\ 136 & 90 & 28 & 13 \\ 201 & 166 & 8 & 28 \end{pmatrix} + 7.I = \begin{pmatrix} 197 & 216 & 148 & 122 \\ 234 & 241 & 102 & 222 \\ 64 & 94 & 251 & 214 \\ 164 & 8 & 90 & 197 \end{pmatrix}$$

$$r_B = \begin{pmatrix} 197 & 216 & 148 & 122 \\ 234 & 241 & 102 & 222 \\ 64 & 94 & 251 & 214 \\ 164 & 8 & 90 & 197 \end{pmatrix}^{12} \begin{pmatrix} 244 & 25 & 67 & 117 \\ 9 & 25 & 0 & 59 \\ 11 & 2 & 195 & 219 \\ 104 & 38 & 121 & 44 \end{pmatrix} \begin{pmatrix} 197 & 216 & 148 & 122 \\ 234 & 241 & 102 & 222 \\ 64 & 94 & 251 & 214 \\ 164 & 8 & 90 & 197 \end{pmatrix}^7 =$$

$$= \begin{pmatrix} 242 & 43 & 53 & 189 \\ 199 & 73 & 72 & 195 \\ 51 & 96 & 51 & 87 \\ 80 & 140 & 255 & 10 \end{pmatrix}$$

Sexto Paso

Alicia calcula su clave compartida

$$K_A = (f(a))^m \cdot r_B \cdot (f(a))^n$$

Es decir

$$K_A = \begin{pmatrix} 101 & 70 & 91 & 29 \\ 163 & 208 & 97 & 193 \\ 111 & 46 & 127 & 167 \\ 230 & 1 & 68 & 106 \end{pmatrix}^{12} \begin{pmatrix} 242 & 43 & 53 & 189 \\ 199 & 73 & 72 & 195 \\ 51 & 96 & 51 & 87 \\ 80 & 140 & 255 & 10 \end{pmatrix} \begin{pmatrix} 101 & 70 & 91 & 29 \\ 163 & 208 & 97 & 193 \\ 111 & 46 & 127 & 167 \\ 230 & 1 & 68 & 106 \end{pmatrix}^7 =$$

$$= \begin{pmatrix} 192 & 211 & 72 & 228 \\ 166 & 175 & 90 & 150 \\ 82 & 221 & 174 & 162 \\ 65 & 136 & 163 & 107 \end{pmatrix}$$

Séptimo Paso

Bernardo calcula su clave compartida

$$K_B = (h(a))^m \cdot r_A \cdot (h(a))^n$$

Entonces

$$K_B = \begin{pmatrix} 197 & 216 & 148 & 122 \\ 234 & 241 & 102 & 222 \\ 64 & 94 & 251 & 214 \\ 164 & 8 & 90 & 197 \end{pmatrix}^{12} \begin{pmatrix} 214 & 107 & 138 & 118 \\ 4 & 243 & 148 & 80 \\ 12 & 233 & 60 & 176 \\ 73 & 202 & 107 & 91 \end{pmatrix} \begin{pmatrix} 197 & 216 & 148 & 122 \\ 234 & 241 & 102 & 222 \\ 64 & 94 & 251 & 214 \\ 164 & 8 & 90 & 197 \end{pmatrix}^7 =$$

$$= \begin{pmatrix} 192 & 211 & 72 & 228 \\ 166 & 175 & 90 & 150 \\ 82 & 221 & 174 & 162 \\ 65 & 136 & 163 & 107 \end{pmatrix}$$

¿Por qué se eligen como claves privadas a polinomios en $\mathbb{Z}_{16}[x]$?

Los coeficientes de los polinomios que constituyen las claves privadas se eligen en $\mathbb{Z}_{16}$ para que cada uno de estos coeficientes puedan ser almacenados en 1 byte, que es la unidad mínima de almacenamiento de datos. De este modo, los polinomios ocuparán la menor cantidad de memoria.

Por otro lado, los grados de los polinomios se seleccionan también en $\mathbb{Z}_{16}$ para lograr un nivel de seguridad computacional aceptable. Para corroborar esta afirmación, probaremos primero la siguiente Proposición.

Proposición 5.15

Sea $m \in \mathbb{N}$. Entonces existen $m^n - 1$ polinomios no nulos en $\mathbb{Z}_m[x]$ de grado menor que n .

Demostración

Sea $f(x)$ un polinomio en $\mathbb{Z}_m[x]$ de grado menor que n . Entonces

$$f(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_{n-1} x^{n-1}$$

Con $a_i \in \mathbb{Z}_m \quad \forall i = 0, 1, \dots, n-1$

Como $a_i \in \mathbb{Z}_m = \{0, 1, \dots, m-1\}$, los coeficientes a_i pueden tomar m valores distintos, $\forall i = 0, 1, \dots, n-1$. Por el Principio de la Multiplicación, y debido a que cualquier polinomio $f(x)$ de este tipo tiene n coeficientes (nulos o no), resulta que existen m^n polinomios con estas características. Al eliminar el polinomio nulo, obtenemos $m^n - 1$ polinomios no nulos en $\mathbb{Z}_m[x]$ de grado menor que n . ■

¿Por qué las claves privadas son polinomios de grado menor que 16?

La seguridad del algoritmo que presentamos radica en la dificultad para hallar una de las claves privadas, ya sea la de Alicia o la de Bernardo. Más precisamente, el “intruso” conoce a, b, m, n, r_A y r_B . Entonces podría ir probando cuál es la función $f(x)$ o $h(x)$ que verifica una de las ecuaciones:

$$r_A = (f(a))^m \cdot b \cdot (f(a))^n ; \quad r_B = (h(a))^m \cdot b \cdot (h(a))^n$$

De este modo, conociendo $f(x)$ o bien $h(x)$, el intruso descubriría la clave, ya que

$$K_A = (f(a))^m \cdot r_B \cdot (f(a))^n \wedge K_B = (h(a))^m \cdot r_A \cdot (h(a))^n$$

Pero ¿qué tan difícil es hallar $f(x)$ o $h(x)$? Pues bien, supongamos que el intruso desea descubrir la clave privada $f(x) \in \mathbb{Z}_{16}[x]$ de Alicia. Entonces, por la Proposición 5.15, habrá $16^n - 1 = 2^{4n} - 1$ posibles polinomios $f(x)$ de grado menor que n . En consecuencia, si el intruso empleara la “fuerza bruta”, debería realizar a lo sumo $2^{4n} - 1$ intentos hasta encontrar $f(x)$.

En particular, tomando $n = 16$, la cantidad de posibles claves privadas que pudo haber elegido Alicia es

$$2^{4n} - 1 = 2^{4(16)} - 1 = 2^{64} - 1 = 18446744073709551615$$

Por lo tanto, utilizando “la fuerza bruta”, encontrar a $f(x)$ sería muy difícil, ya que la cantidad de funciones que verifican las hipótesis del algoritmo es sorprendentemente numerosa. Este hecho pone de manifiesto la seguridad criptográfica del modelo.

Análisis del tiempo y espacio de memoria requerido para la ejecución del Algoritmo de Diffie-Hellman aplicando matrices

Primeramente destacaremos que, cuando se analiza el tiempo de ejecución de un determinado algoritmo, se cuentan solamente la cantidad de multiplicaciones y divisiones numéricas, ya que el tiempo que tarda el ordenador en efectuar una multiplicación o división es predominantemente mayor al tiempo requerido para efectuar una suma o resta. Por esta razón sólo se contarán las multiplicaciones que lleva a cabo el algoritmo, ya que no hay divisiones.

Por otro lado, según lo expuesto en el Ejemplo 2.24, cuando se multiplican o suman dos enteros no negativos que ocupan 1 byte cada uno, el resultado es almacenado en 2 bytes. Antes de iniciar el proceso de conteo, mostraremos el procedimiento que utiliza el ordenador para efectuar algunas operaciones que serán aplicadas en el algoritmo de Diffie-Hellman con matrices.

Cálculo del resto de la división entera

Dado un entero no negativo num de 2 bytes, el pseudocódigo que calcula el resto de la división entera de num por 256 es el siguiente:

```
Mientras  $num \geq 256$  hacer
     $num := num - 256$ 
Fin Mientras
Resto: =  $num$ 
```

El resto de la división entera quedará almacenado en la variable $Resto$, que será un número comprendido entre 0 y 255. Por lo tanto, a la variable $Resto$ le asignaremos 1 byte de memoria.

Cálculo del producto modular de dos matrices

Dadas dos matrices $A, B \in M_4[\mathbb{Z}_{256}]$, su producto $A \cdot B \pmod{256}$ es una matriz $C \in M_4[\mathbb{Z}_{256}]$ de modo tal que sus coeficientes c_{ij} verifican

$$c_{ij} = \sum_{k=1}^4 a_{ik} \cdot b_{kj} \pmod{256}, \forall i, j=1,2,3,4$$

La siguiente Proposición muestra la cantidad de multiplicaciones y el espacio de memoria requerido para efectuar el producto de estas dos matrices.

Proposición 5.16

Sean $A, B \in M_4[\mathbb{Z}_{256}]$. Entonces, al calcular $A \cdot B \pmod{256}$, se efectúan 64 multiplicaciones y se requieren 27 bytes de memoria.

Demostración

El pseudocódigo que calcula la matriz $C = A \cdot B \pmod{256}$ es el siguiente

```
Para i = 1 hasta 4 hacer
  Para j = 1 hasta 4 hacer
    Suma := 0
    Para k = 1 hasta 4 hacer
      Producto :=  $a_{i k} \cdot b_{k j}$ 
      // Calculamos el resto de dividir Producto por 256 //
      Mientras Producto  $\geq 256$  hacer
        Producto := Producto - 256
      Fin Mientras
      RestoProducto := Producto
      Sumaux := Suma + RestoProducto
      // Calculamos el resto de dividir Sumaux por 256 //
      Mientras Sumaux  $\geq 256$  hacer
        Sumaux := Sumaux - 256
      Fin Mientras
      Suma := Sumaux
    Fin Para
     $c_{i j} := Suma$ 
  Fin Para
Fin Para
```

Cabe recordar que cada coeficiente de las matrices A y B ocupa 1 byte de memoria. Además, en este proceso de conteo no tendremos en cuenta el espacio ocupado por las matrices A y B , ya que suponemos que han sido previamente cargadas. Observando el pseudocódigo del programa, iremos contando el espacio de memoria que precisa cada variable:

Espacio de memoria requerido

Variable *Suma*: 1 byte

Variable *Producto*: 2 bytes

(Pues a_{ik} y b_{kj} ocupa cada uno 1 byte. Luego el producto de ambos es almacenado en 2 bytes)

Constante 256: 2 bytes

Variable *RestoProducto*: 1 byte

Variable *Sumaux*: 2 bytes

(Pues *Suma* y *RestoProducto* ocupa cada uno 1 byte. Luego la suma de ambos es almacenada en 2 bytes)

Variables c_{ij} , con $i, j = 1, 2, 3, 4$: 16 bytes

Variables contadores i, j, k : 3 bytes

Por lo tanto, el espacio total de memoria necesario es:

$(1 + 2 + 2 + 1 + 2 + 16 + 3)$ bytes = 27 bytes

Cantidad de multiplicaciones efectuadas

En el pseudocódigo vemos que sólo hay, para cada i, j y k , una multiplicación en la línea 5. Luego, el total de multiplicaciones efectuadas es

$$\sum_{i=1}^4 \sum_{j=1}^4 \sum_{k=1}^4 1 = 64 .$$

■

En la siguiente Proposición deduciremos el espacio de memoria requerido para efectuar la suma modular de matrices.

Proposición 5.17

Sean $A, B \in M_4[\mathbb{Z}_{256}]$. Entonces, para calcular $A + B \pmod{256}$ se requieren 22 bytes de memoria.

Demostración

El pseudocódigo que calcula la matriz $C = A + B \pmod{256}$ es:

```
Para i = 1 hasta 4 hacer
  Para j = 1 hasta 4 hacer
    Suma :=  $a_{ij} + b_{ij}$ 
    // Calculamos el resto de dividir Suma por 256 //
    Mientras Suma  $\geq 256$  hacer
      Suma := Suma - 256
    Fin Mientras
     $c_{ij} :=$  Suma
  Fin Para
Fin Para
```

Espacio de memoria requerido

Variable Suma: 2 bytes

(Pues a_{ij} y b_{ij} ocupa cada uno 1 byte. Luego la suma de ambos es almacenada en 2 bytes)

Constante 256: 2 bytes

Variables c_{ij} , con $i, j = 1, 2, 3, 4$: 16 bytes

Variables contadores i, j : 2 bytes

Por lo tanto, el espacio total de memoria necesario es:

$$(2 + 2 + 16 + 2) \text{ bytes} = 22 \text{ bytes} .$$

■

Cómo valuar un polinomio: El Algoritmo de Horner

El Algoritmo de Horner, llamado así en honor a William Horner (1789-1837) es un método que permite evaluar de forma eficiente un polinomio. Para entender cómo funciona, observemos que un polinomio puede expresarse del siguiente modo:

Polinomio de grado 2:

$$p(x) = a_0 + a_1x + a_2x^2 = a_0 + x(a_1 + a_2x)$$

Polinomio de grado 3:

$$\begin{aligned} p(x) &= a_0 + a_1x + a_2x^2 + a_3x^3 = a_0 + x(a_1 + a_2x + a_3x^2) = \\ &= a_0 + x(a_1 + x(a_2 + a_3x)) \end{aligned}$$

Polinomio de grado 4:

$$\begin{aligned} p(x) &= a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 = a_0 + x(a_1 + a_2x + a_3x^2 + a_4x^3) = \\ &= a_0 + x(a_1 + x(a_2 + a_3x + a_4x^2)) = a_0 + x(a_1 + x(a_2 + x(a_3 + a_4x))) \end{aligned}$$

En general, un polinomio de grado n puede expresarse así:

$$p(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n = a_0 + x(a_1 + x(a_2 + \dots + x(a_{n-1} + a_nx)))$$

Para valuar un polinomio en $x = x_0$, comenzamos a efectuar las operaciones aritméticas que se encuentran en el paréntesis más interno, es decir, los cálculos se llevan a cabo de adentro hacia afuera. El pseudocódigo que implementa este algoritmo es el siguiente:

```
Resultado := an

Para i = 1 hasta n hacer
    Resultado := an-i + Resultado . x0
Fin Para
```

Así, el valor $p(x_0)$ quedará almacenado en la variable *Resultado*. En la siguiente Proposición calcularemos la cantidad de sumas y multiplicaciones que efectúa el Algoritmo de Horner y lo compararemos con el Método Clásico, que es el que resulta de calcular las potencias mediante la repetición de multiplicaciones.

Proposición 5.18

Sea $p(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$ un polinomio de grado n . Entonces

- 1) Al evaluar $p(x_0)$, el Algoritmo de Horner efectúa n sumas y n multiplicaciones.
- 2) Al evaluar $p(x_0)$, el Método Clásico realiza n sumas y $\frac{n^2 + n}{2}$ multiplicaciones.

Demostración

1) Si observamos el pseudocódigo del Algoritmo de Horner, comprobamos que sólo hay, para cada i , una suma y una multiplicación en la línea 3. Por lo tanto

$$\text{Total de multiplicaciones} = \text{Total de sumas} = \sum_{i=1}^n 1 = n.$$

2) En el Método Clásico resolvemos primero cada término y luego los sumamos.

Como en el i -ésimo término se efectúan i multiplicaciones para $i = 0, 1, \dots, n$ entonces

$$\text{Total de multiplicaciones: } \sum_{i=0}^n i = \frac{n(n+1)}{2} = \frac{n^2 + n}{2}.$$

Y claramente

Total de sumas: n . ■

Como podemos comprobar, y de acuerdo a la Definición 2.29, en el Algoritmo de Horner se ejecutan $O(n)$ multiplicaciones, mientras que con el Método Clásico de valuación se realizan $O(n^2)$ productos. Este hecho prueba la eficiencia del Algoritmo de Horner, ya que el tiempo que se tarda en evaluar un polinomio con este método es significativamente menor que el del Método Clásico.

A continuación calcularemos la cantidad de multiplicaciones y el espacio requerido para evaluar un polinomio, cuando la variable x es una matriz del conjunto $M_4[\mathbb{Z}_{256}]$.

Proposición 5.19

Sea $p(x) = a_0 + a_1x + \dots + a_nx^n$ un polinomio de grado n y $A \in M_4[\mathbb{Z}_{256}]$.

Entonces

- 1) Al calcular $p(A) \pmod{256}$ aplicando el Algoritmo de Horner, se realizan $64n$ multiplicaciones.
- 2) Para calcular $p(A) \pmod{256}$ con el Algoritmo de Horner, se requieren 82 bytes de memoria.

Demostración

1) El pseudocódigo del Algoritmo de Horner aplicando las operaciones modulares de matrices es el siguiente:

```
Resultado :=  $a_n \cdot I$ 
Para  $i = 1$  hasta  $n$  hacer
    Resultado :=  $(a_{n-i} \cdot I + \text{Resultado} \cdot A) \pmod{256}$ 
Fin Para
```

Donde I es la matriz identidad de orden 4.

Observamos que, para cada i , sólo hay un producto matricial modular, el cual proviene del cálculo de $\text{Resultado} \cdot A \pmod{256}$. Los productos matriciales $a_n \cdot I$ y $a_{n-i} \cdot I$ no se consideran, pues para obtenerlos no se necesita efectuar ninguna multiplicación. Por la Proposición 5.16, cada producto matricial modular efectúa 64 multiplicaciones. En consecuencia, la cantidad total de multiplicaciones efectuadas es

$$\sum_{i=1}^n 64 = 64n.$$

2) De acuerdo al pseudocódigo, los espacios de memoria necesarios son:

Matriz *Resultado*: 16 bytes

Matriz $(\text{Resultado} \cdot A) \pmod{256}$: 27 bytes

(Por la Proposición 5.16)

Matriz $a_{n-i} \cdot I$: 16 bytes

Matriz $(a_{n-i} \cdot I + \text{Resultado} \cdot A) \pmod{256}$: 22 bytes

(Por la Proposición 5.17)

Variable contador i : 1 byte

Por lo tanto, el espacio total de memoria requerido es:

$$(16 + 27 + 16 + 22 + 1) \text{ bytes} = 82 \text{ bytes}$$

■

A continuación analizaremos el tiempo y el espacio de memoria que se precisa para que Alicia y Bernardo generen la clave que compartirán. Luego lo compararemos con el Algoritmo de Diffie-Hellman clásico.

Análisis del tiempo requerido para ejecutar el Algoritmo de Diffie-Hellman aplicando matrices

En el siguiente cuadro mostramos, paso a paso, la cantidad de operaciones que lleva a cabo el algoritmo.

Paso	Operación a realizar	Cantidad de multiplicaciones
1	Alicia elige dos matrices no nulas, $a, b \in M_4[\mathbb{Z}_{256}]$, y dos números no nulos m y $n \in \mathbb{Z}_{16}$	
2	Alicia elige un polinomio no nulo $f(x) \in \mathbb{Z}_{16}[x]$ de grado $k \leq 15$.	
3	Bernardo elige un polinomio no nulo $h(x) \in \mathbb{Z}_{16}[x]$ de grado $q \leq 15$.	
4	Alicia calcula $r_A = (f(a))^m \cdot b \cdot (f(a))^n$	$64 \cdot (m + n + k - 1)$
5	Bernardo computa $r_B = (h(a))^m \cdot b \cdot (h(a))^n$	$64 \cdot (m + n + q - 1)$
6	Alicia calcula su clave compartida $K_A = (f(a))^m \cdot r_B \cdot (f(a))^n$	128
7	Bernardo calcula su clave compartida $K_B = (h(a))^m \cdot r_A \cdot (h(a))^n$	128

Detalles de cada Paso

Paso 4

Por la Proposición 5.19, al calcular de $c = f(a)$ se realizan $64k$ multiplicaciones.

Por otro lado, en el cálculo de $c^m \cdot b \cdot c^n$ se efectúan $(m + n - 1)$ multiplicaciones matriciales. Por la Proposición 5.16, al calcular un producto matricial modular se realizan 64 multiplicaciones. Luego, para calcular $c^m \cdot b \cdot c^n$ se llevan a cabo $64 \cdot (m + n - 1)$ multiplicaciones. En consecuencia, al calcular $r_A = (f(a))^m \cdot b \cdot (f(a))^n$ se efectúan $64k + 64 \cdot (m + n - 1) = 64 \cdot (m + n + k - 1)$ multiplicaciones.

Paso 5

Es similar al paso anterior y se obtienen $64 \cdot (m + n + q - 1)$ multiplicaciones.

Paso 6

Cuando Alicia calcula su clave $K_A = (f(a))^m \cdot r_B \cdot (f(a))^n$, las matrices $(f(a))^m$ y $(f(a))^n$ ya están computadas. Luego, al calcular K_A se realizan 2 multiplicaciones matriciales. Luego, por la Proposición 5.16, al computar se llevan a cabo $2 \cdot (64) = 128$ multiplicaciones.

Paso 7

Es similar al paso anterior y se obtienen 128 multiplicaciones.

En consecuencia,

Total de operaciones efectuadas en la computadora de Alicia:

$$64 \cdot (m + n + k - 1) + 128 \text{ multiplicaciones}$$

Donde

k : es el grado del polinomio $f(x)$, la clave privada de Alicia.

m, n : son números pertenecientes a $\mathbb{Z}_{16}$ que Alicia eligió.

Total de operaciones efectuadas en la computadora de Bernardo:

$$64 \cdot (m + n + q - 1) + 128 \text{ multiplicaciones}$$

Donde

q : es el grado del polinomio $h(x)$, la clave privada de Bernardo.

m, n : son números pertenecientes a $\mathbb{Z}_{16}$ que Alicia eligió.

Como podemos comprobar, la cantidad de operaciones efectuadas tanto en la computadora de Alicia como en la de Bernardo es $O(k)$ y $O(q)$ respectivamente. Es decir, es lineal en términos del grado del polinomio seleccionado como clave privada. La situación extrema se produce cuando $k=q=m=n=15$. En este caso, los procesadores de Alicia y de Bernardo efectuarán 2944 multiplicaciones para generar la clave que compartirán.

Comparación con el Algoritmo de Diffie-Hellman clásico.

Según lo desarrollado en la tesis de Cecilia Prieto ([4], pág.95), si a y b son los números que eligen Alicia y Bernardo respectivamente, al aplicar el Algoritmo de Diffie-Hellman clásico, entonces

Total de operaciones efectuadas en la computadora de Alicia:

$$6 L_a - 4$$

Donde

L_a : es la cantidad de dígitos binarios que tiene el número a .

Total de operaciones efectuadas en la computadora de Bernardo:

$$6 L_b - 4$$

Donde

L_b : es la cantidad de dígitos binarios que tiene el número b .

Teniendo en cuenta que tanto a como b son números naturales muy grandes y por lo tanto, tienen numerosos dígitos binarios, el Algoritmo de Diffie-Hellman empleando matrices efectúa, en general, un número de operaciones menor que el de Diffie-Hellman clásico.

Ejemplo 5.20

Alicia piensa utilizar el protocolo de Diffie-Hellman con matrices para generar una clave secreta. Para ello pretende seleccionar los números $m = n = 1$, y como clave privada a un polinomio de grado $k=1$. Si en su lugar aplicara el Algoritmo de Diffie-Hellman clásico, ¿cuál es el mayor valor que podrá tener el número a para que tarde menos tiempo en generar una clave que empleando Diffie-Hellman con matrices?

Solución

Para que el Algoritmo de Diffie-Hellman clásico tarde menos tiempo que el protocolo de Diffie-Hellman con matrices se debe verificar que

$$6 L_a - 4 \leq 64.(m + n + k - 1) + 128 .$$

Es decir,

$$6 L_a - 4 \leq 64.(2) + 128 \Rightarrow 6 L_a - 4 \leq 256 \Rightarrow L_a \leq 43.33 .$$

Por lo tanto, el número a deberá tener a lo sumo 43 dígitos binarios. El mayor número de 43 dígitos binarios es aquél que tiene 43 dígitos iguales a 1. Dicho número en base 10 es

$$\sum_{i=0}^{42} 2^i = \frac{2^{43} - 1}{2 - 1} = 2^{43} - 1 = 8796093022207 .$$

En consecuencia,

$$a \leq 8796093022207.$$

Análisis del espacio de memoria requerido para la ejecución del Algoritmo de Diffie-Hellman aplicando matrices

Para este análisis recordaremos que un número del conjunto $\mathbb{Z}_{16}$ ocupa 1 byte y que una matriz perteneciente a $M_4[\mathbb{Z}_{256}]$ abarca 16 bytes de memoria.

En base a estas observaciones, en los siguientes cuadros mostramos la cantidad de memoria que requiere el algoritmo.

Para la computadora de Alicia

Elemento	Cantidad de bytes requeridos
$a, b \in M_4[\mathbb{Z}_{256}]$	16 bytes + 16 bytes = 32 bytes
m y $n \in \mathbb{Z}_{16}$	1 byte + 1 byte = 2 bytes
$f(x) \in \mathbb{Z}_{16}[x]$ de grado $k \leq 15$.	$(k+1)$ bytes
$f(a)$	16 bytes
$(f(a))^m$	16 bytes
$(f(a))^n$	16 bytes
r_A	16 bytes
r_B	16 bytes
K_A	16 bytes

Además, teniendo en cuenta las Proposiciones 5.16 y 5.19, los siguientes espacios de memoria son utilizados para el cómputo de los elementos de la tabla anterior:

Espacio para el cómputo de:	Cantidad de bytes requeridos
$f(a)$	82 bytes
Producto matricial modular	27 bytes

Por lo tanto,

Total de espacio de memoria requerido en la computadora de Alicia:

$$(32 + 2 + k + 1 + 6 \cdot (16) + 82 + 27) \text{ bytes} = (240 + k) \text{ bytes}$$

Donde k : es el grado del polinomio $f(x)$, la clave privada de Alicia.

Para la computadora de Bernardo

El procedimiento de cálculo es igual al efectuado en la computadora de Alicia. De este modo,

Total de espacio de memoria requerido en la computadora de Bernardo:

$$(32 + 2 + q + 1 + 6 \cdot (16) + 82 + 27) \text{ bytes} = (240 + q) \text{ bytes}$$

Donde q : es el grado del polinomio $h(x)$, la clave privada de Bernardo.

Comparación con el Algoritmo de Diffie-Hellman clásico.

Según lo desarrollado en la tesis de Cecilia Prieto ([4], pág.91), si p es un número primo muy grande que Alicia y Bernardo acuerdan utilizar, entonces

Total de espacio de memoria en la computadora de Alicia:

$$12 L_p \text{ bits}$$

Total de espacio de memoria en la computadora de Bernardo:

$$12 L_p \text{ bits}$$

Donde L_p : es la cantidad de dígitos binarios que tiene el número p .

Nuevamente, si se tiene en cuenta que p es un número primo muy grande y por lo tanto, tiene numerosos dígitos binarios, el Algoritmo de Diffie-Hellman empleando matrices necesita mucho menos espacio de memoria que el de Diffie-Hellman clásico.

Consideraciones acerca de la seguridad computacional del protocolo propuesto

Debemos resaltar que incrementar el valor de los parámetros m , n o de las matrices a , b no aporta seguridad al protocolo, dado que son elementos públicos. La seguridad reside exclusivamente en el cardinal del conjunto de las claves privadas, es decir, en el grado y en los coeficientes de los polinomios $f(x)$ y $g(x)$ que elijan Alicia y Bernardo. Como vimos, el cardinal de dicho conjunto es

$$2^{64} - 1 = 18446744073709551615$$

Si el intruso empleara “la fuerza bruta” para descubrir una de las claves privadas, y si los cálculos del ataque se hicieran a razón de 10^6 tanteos por segundo, podría tardar hasta 584942 000000 años en encontrarla.

Por otro lado, si el intruso tratara, también con “la fuerza bruta”, de hallar la clave secreta $K_A = K_B$ entonces, por tratarse de una matriz del conjunto $M_4[\mathbb{Z}_{256}]$, la cantidad de posibles claves no nulas es, aplicando el Principio de la Multiplicación:

$$256^{16} - 1 = (2^8)^{16} - 1 = 2^{128} - 1 = 340282366920938463463374607431768211455$$

Con lo cual podría tardar hasta $(1.07903)10^{31}$ años en descubrirla.

Finalizaremos este capítulo mostrando el código fuente del programa que implementa el Algoritmo de Diffie-Hellman aplicando matrices. El mismo fue diseñado mediante el software Wolfram Mathematica 10.0.

Implementación del Algoritmo de Diffie-Hellman con matrices aplicando Wolfram Mathematica

El siguiente programa, diseñado mediante el software Wolfram Mathematica 10.0, permite generar una clave secreta que compartirán dos personas, aplicando el protocolo de Diffie-Hellman con matrices. El programa pide ingresar, en un cuadro de diálogo, los números m , n , las matrices a , b y los coeficientes de los polinomios $f(x)$ y $g(x)$, como mostramos en la figura:

Ingrese dos números enteros positivos $\in \mathbb{Z}_{16}$:

$m =$

$n =$

Ingrese los coeficientes de la matriz A de orden 4:

$A = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$

Ingrese los coeficientes de la matriz B de orden 4:

$B = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$

Ingrese los coeficientes del polinomio $f(x) \in \mathbb{Z}_{16}[x]$ de Alice:

$f(x) = ($ $+$ x $+$ x^2 $+$ x^3 $+$ x^4 $+$ x^5 $+$ x^6 $+$ x^7 $+$ x^8 $+$ x^9 $+$ x^{10} $+$ x^{11} $+$ x^{12} $+$ x^{13} $+$ x^{14} $+$ x^{15} $)$

Ingrese los coeficientes del polinomio $h(x) \in \mathbb{Z}_{16}[x]$ de Bob:

$h(x) = ($ $+$ x $+$ x^2 $+$ x^3 $+$ x^4 $+$ x^5 $+$ x^6 $+$ x^7 $+$ x^8 $+$ x^9 $+$ x^{10} $+$ x^{11} $+$ x^{12} $+$ x^{13} $+$ x^{14} $+$ x^{15} $)$

Figura 5.2: Ingreso por pantalla de los datos para generar una clave mediante el protocolo Diffie-Hellman con matrices.

Ingresaremos los mismos datos que figuran en el ejemplo que expusimos. Al ejecutar el programa, se muestra el siguiente resultado:

La clave de sesión compartida $K_A \bmod 256$ es:

$\begin{pmatrix} 192 & 211 & 72 & 228 \\ 166 & 175 & 90 & 150 \\ 82 & 221 & 174 & 162 \\ 65 & 136 & 163 & 107 \end{pmatrix}$

La clave de sesión compartida $K_B \bmod 256$ es:

$\begin{pmatrix} 192 & 211 & 72 & 228 \\ 166 & 175 & 90 & 150 \\ 82 & 221 & 174 & 162 \\ 65 & 136 & 163 & 107 \end{pmatrix}$

El código fuente del programa lo mostramos a continuación.

Código fuente del programa que genera una clave aplicando el protocolo de Diffie-Hellman con matrices

```
Panel[Grid[{
  {Style["Ingrese dos números enteros positivos  $\in \mathbb{Z}_{16}$ :", Bold], SpanFromLeft}, {"m=", InputField[Dynamic[m], FieldSize -> 2]},
  {"n=", InputField[Dynamic[n], FieldSize -> 2]}},
{Style["Ingrese los coeficientes de la matriz A de orden 4:", Bold], SpanFromLeft},
{"A=",
  MatrixForm[{{InputField[Dynamic[a11], FieldSize -> 2], InputField[Dynamic[a12], FieldSize -> 2],
    InputField[Dynamic[a13], FieldSize -> 2], InputField[Dynamic[a14], FieldSize -> 2]},
  {InputField[Dynamic[a21], FieldSize -> 2], InputField[Dynamic[a22], FieldSize -> 2], InputField[Dynamic[a23], FieldSize -> 2],
    InputField[Dynamic[a24], FieldSize -> 2]}, {InputField[Dynamic[a31], FieldSize -> 2], InputField[Dynamic[a32], FieldSize -> 2],
    InputField[Dynamic[a33], FieldSize -> 2], InputField[Dynamic[a34], FieldSize -> 2]},
  {InputField[Dynamic[a41], FieldSize -> 2], InputField[Dynamic[a42], FieldSize -> 2], InputField[Dynamic[a43], FieldSize -> 2],
    InputField[Dynamic[a44], FieldSize -> 2]}}}],

Panel[Grid[{
  {Style["Ingrese dos números enteros positivos  $\in \mathbb{Z}_{16}$ :", Bold], SpanFromLeft}, {"m=", InputField[Dynamic[m], FieldSize -> 2]},
  {"n=", InputField[Dynamic[n], FieldSize -> 2]}},
{Style["Ingrese los coeficientes de la matriz A de orden 4:", Bold], SpanFromLeft},
{"A=",
  MatrixForm[{{InputField[Dynamic[a11], FieldSize -> 2], InputField[Dynamic[a12], FieldSize -> 2],
    InputField[Dynamic[a13], FieldSize -> 2], InputField[Dynamic[a14], FieldSize -> 2]},
  {InputField[Dynamic[a21], FieldSize -> 2], InputField[Dynamic[a22], FieldSize -> 2], InputField[Dynamic[a23], FieldSize -> 2],
    InputField[Dynamic[a24], FieldSize -> 2]}, {InputField[Dynamic[a31], FieldSize -> 2], InputField[Dynamic[a32], FieldSize -> 2],
    InputField[Dynamic[a33], FieldSize -> 2], InputField[Dynamic[a34], FieldSize -> 2]},
  {InputField[Dynamic[a41], FieldSize -> 2], InputField[Dynamic[a42], FieldSize -> 2], InputField[Dynamic[a43], FieldSize -> 2],
    InputField[Dynamic[a44], FieldSize -> 2]}}}],

{Style["Ingrese los coeficientes de la matriz B de orden 4:", Bold], SpanFromLeft},
{"B=",
  MatrixForm[{{InputField[Dynamic[b11], FieldSize -> 2], InputField[Dynamic[b12], FieldSize -> 2], InputField[Dynamic[b13], FieldSize -> 2],
    InputField[Dynamic[b14], FieldSize -> 2]}, {InputField[Dynamic[b21], FieldSize -> 2], InputField[Dynamic[b22], FieldSize -> 2],
    InputField[Dynamic[b23], FieldSize -> 2], InputField[Dynamic[b24], FieldSize -> 2]},
  {InputField[Dynamic[b31], FieldSize -> 2], InputField[Dynamic[b32], FieldSize -> 2], InputField[Dynamic[b33], FieldSize -> 2],
    InputField[Dynamic[b34], FieldSize -> 2]}, {InputField[Dynamic[b41], FieldSize -> 2], InputField[Dynamic[b42], FieldSize -> 2],
    InputField[Dynamic[b43], FieldSize -> 2], InputField[Dynamic[b44], FieldSize -> 2]}}}],
{Style["Ingrese los coeficientes del polinomio  $f(x) \in \mathbb{Z}_{16}[x]$  de Alice:", Bold], SpanFromLeft},
{"f(x)=",
  MatrixForm[{{InputField[Dynamic[f0], FieldSize -> 2], "+", InputField[Dynamic[f1], FieldSize -> 2], "x +",
    InputField[Dynamic[f2], FieldSize -> 2], "x^2 +", InputField[Dynamic[f3], FieldSize -> 2], "x^3 +",
    InputField[Dynamic[f4], FieldSize -> 2], "x^4 +", InputField[Dynamic[f5], FieldSize -> 2], "x^5 +",
    InputField[Dynamic[f6], FieldSize -> 2], "x^6 +", InputField[Dynamic[f7], FieldSize -> 2], "x^7 +",
    InputField[Dynamic[f8], FieldSize -> 2], "x^8 +", InputField[Dynamic[f9], FieldSize -> 2], "x^9 +",
    InputField[Dynamic[f10], FieldSize -> 2], "x^10 +", InputField[Dynamic[f11], FieldSize -> 2], "x^11 +",
    InputField[Dynamic[f12], FieldSize -> 2], "x^12 +", InputField[Dynamic[f13], FieldSize -> 2], "x^13 +",
    InputField[Dynamic[f14], FieldSize -> 2], "x^14 +", InputField[Dynamic[f15], FieldSize -> 2], "x^15"}}, TableSpacing -> {None, 0.1}]],

{Style["Ingrese los coeficientes del polinomio  $h(x) \in \mathbb{Z}_{16}[x]$  de Bob:", Bold], SpanFromLeft},
{"h(x)=",
  MatrixForm[{{InputField[Dynamic[h0], FieldSize -> 2], "+", InputField[Dynamic[h1], FieldSize -> 2], "x +",
    InputField[Dynamic[h2], FieldSize -> 2], "x^2 +", InputField[Dynamic[h3], FieldSize -> 2], "x^3 +",
    InputField[Dynamic[h4], FieldSize -> 2], "x^4 +", InputField[Dynamic[h5], FieldSize -> 2], "x^5 +",
    InputField[Dynamic[h6], FieldSize -> 2], "x^6 +", InputField[Dynamic[h7], FieldSize -> 2], "x^7 +",
    InputField[Dynamic[h8], FieldSize -> 2], "x^8 +", InputField[Dynamic[h9], FieldSize -> 2], "x^9 +",
    InputField[Dynamic[h10], FieldSize -> 2], "x^10 +", InputField[Dynamic[h11], FieldSize -> 2], "x^11 +",
    InputField[Dynamic[h12], FieldSize -> 2], "x^12 +", InputField[Dynamic[h13], FieldSize -> 2], "x^13 +",
    InputField[Dynamic[h14], FieldSize -> 2], "x^14 +", InputField[Dynamic[h15], FieldSize -> 2], "x^15"}}, TableSpacing -> {None, 0.1}]]
}, Spacings -> {0, 1}]]],
```

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix};$$

(*En las matrices Suma y SumaB se guardará el valor de f(A) y h(A) respectivamente *)

$$Suma = \begin{pmatrix} f_0 & 0 & 0 & 0 \\ 0 & f_0 & 0 & 0 \\ 0 & 0 & f_0 & 0 \\ 0 & 0 & 0 & f_0 \end{pmatrix};$$

$$SumaB = \begin{pmatrix} h_0 & 0 & 0 & 0 \\ 0 & h_0 & 0 & 0 \\ 0 & 0 & h_0 & 0 \\ 0 & 0 & 0 & h_0 \end{pmatrix};$$

(* En los vectores f y h guardaremos los coeficientes de los polinomios f(x) y h(x) respectivamente *)

f = {f0, f1, f2, f3, f4, f5, f6, f7, f8, f9, f10, f11, f12, f13, f14, f15};

h = {h0, h1, h2, h3, h4, h5, h6, h7, h8, h9, h10, h11, h12, h13, h14, h15};

(* Ahora calculamos f(A) y lo guardamos en la matriz SUMA *)

```
For[ k = 2, k ≤ 16, k++,
  If[f[[k]] ≠ 0,
    Prod = A;
    For[Potencia = 1, Potencia ≤ k - 2, Potencia++,
      Prod = Mod[Prod.A, 256]
    ];
    Prod = Mod[f[[k]] * Prod, 256];

    Suma = Mod[Prod + Suma, 256];
  ]
]
```

Print["La matriz f(A) mod 256 es:"];

Print[Suma // MatrixForm];

$$B = \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{pmatrix};$$

(* Ahora calculamos R_A *)

```
Potm = Suma;
For[k = 2, k ≤ m, k++,
  Potm = Mod[Potm . Suma, 256]
];
Potn = Suma;
For[k = 2, k ≤ n, k++,
  Potn = Mod[Potn . Suma, 256]
];
```

RA = Mod[Potm . B . Potn, 256];

Print["La matriz R_A mod 256 es:"];

Print[" R_A = ", RA // MatrixForm];

(* Ahora calculamos h(A) y lo guardamos en la matriz SUMAB *)

```
For[ k = 2, k ≤ 16, k++,
  If[h[[k]] ≠ 0,
    Prod = A;
    For[Potencia = 1, Potencia ≤ k - 2, Potencia++,
      Prod = Mod[Prod.A, 256]
    ];
    Prod = Mod[h[[k]] * Prod, 256];
    SumaB = Mod[Prod + SumaB, 256];
  ]
]
```

```

Print["La matriz  $h(A) \bmod 256$  es:"];
Print[SumaB // MatrixForm];
(* Ahora calculamos  $R_B$  *)

PotmB = SumaB;
For[k = 2, k ≤ m, k++,
  PotmB = Mod[PotmB . SumaB, 256]
];
PotnB = SumaB;
For[k = 2, k ≤ n, k++,
  PotnB = Mod[PotnB . SumaB, 256]
];
RB = Mod[PotmB . B . PotnB, 256];
Print["La matriz  $R_B \bmod 256$  es:"];
Print[RB // MatrixForm];
(* Ahora calculamos  $K_A$  y  $K_B$  *)
KA = Mod[Potm . RB . Potn, 256];
Print["La clave de sesión compartida  $K_A \bmod 256$  es:"];
Print[KA // MatrixForm];
KB = Mod[PotmB . RA . PotnB, 256];
Print["La clave de sesión compartida  $K_B \bmod 256$  es:"];
Print[KB // MatrixForm];
]

```

Capítulo 6

Conclusiones

Capítulo 6: Conclusiones

Todas las industrias disponen de sistemas de control y redes de mando diseñados para supervisar y ejecutar sus procesos. Los mismos se aplican, por ejemplo, para controlar procesos industriales a distancia (apertura/cierre de una bomba o entrada/salida de un dato). También son empleados para automatizar procesos electromecánicos, tales como el control de las maquinarias en líneas de montaje.

Este tipo de sistemas utiliza dispositivos de capacidad limitada, a fin de reducir las tareas de mantenimiento y maximizar los costos de operación. Sin embargo, para evitar ser vulnerados, necesitan de algún algoritmo criptográfico que garantice su seguridad.

En la actualidad, la mayoría de los mecanismos de criptografía basan su seguridad en la elevada complejidad computacional requerida para el cálculo de logaritmos discretos y factorización de números enteros. Además, precisan trabajar con aritmética de precisión extendida, es decir, de al menos 4 bytes. Así, algoritmos criptográficos como RSA con claves de 2048 bits de tamaño se consideran seguros, pero resulta imposible aplicarlos en dispositivos con bajo poder de cómputo.

El Algoritmo de Diffie-Hellman aplicado a matrices que desarrollamos en este trabajo, es de los primeros protocolos criptográficos asimétricos basados en la utilización de estructuras algebraicas no conmutativas. Está diseñado especialmente para ser operado en plataformas computacionales pobres, es decir, que no requiera el uso de bibliotecas de precisión extendida, ya que las claves de seguridad pueden generarse usando aritmética entera de 2 bytes.

Además, su seguridad criptográfica está garantizada, debido a la gran dificultad que se presenta a la hora de intentar descubrir las claves privadas.

El uso de este algoritmo que desarrollamos en este trabajo, permitirá que procesadores pequeños puedan implementar métodos criptográficos seguros y así evitar ataques informáticos indeseables.

Referencias

- [1] Monica Sandoval. “*Introducción a la Criptografía*”. Ed. Universidad Tecnológica de Chile, 2013
- [2] Hecht, P. ”*Un modelo compacto de Criptografía Asimétrica empleando Anillos no Conmutativos*”. VII Congreso de Seguridad Informática CIBSI. Panamá, 2013.
- [3] Granado Peralta, S. “*Matemática Discreta*”. Ed. CEIT. Buenos Aires, 2002.
- [4] Prieto, Cecilia Andrea. “*Modelo compacto del algoritmo Diffie-Hellman utilizando cuaterniones*”. Tesis de Licenciatura en Matemática. Directora: Abdel Masih, Samira. Universidad Abierta Interamericana. Buenos Aires, 2019.
- [5] Gibran Grandos Paredes. “*Introducción a la Criptografía*”. Ed. Revista Digital Universitaria, Universidad Nacional Autónoma de México, 2006.
- [6] Menezes J. Alfred. “*Handbook of Applied Cryptography*”. CRC Press, 1997.
- [7] Rojo, Armando. “*Álgebra I*”. Ed. El Ateneo. Buenos Aires, 1982.
- [8] Dorronsoro, J. & Fernández, E. “*Números, Grupos y Anillos*”. Ed. Addison Wesley Iberoamericana. Madrid, 1997.