UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

TESIS

ANALISIS DE LAS VENTAJAS DE LA CRIPTOGRAFIA DE CURVA ELIPTICA, FRENTE A LOS SISTEMAS CRIPTOGRAFICOS ASIMETRICOS ACTUALES

QUE PARA OBTENER EL TÍTULO DE INGENIERO EN COMPUTACIÓN

PRESENTA:

ROJAS JIMÉNEZ GUSTAVO

DIRECTOR DE TESIS:

DR. JOSÉ ABEL HERRERA CAMACHO

MÉXICO, D.F **AGOSTO 2012**

A la patria, que me obligó a superarme.

Agradecimientos

Primordialmente quiero agradecer a la Universidad Nacional Autónoma de México, que me permitió el honor de ser uno de sus hijos, y con quien eternamente estaré agradecido por haberme formado en sus aulas y despertar en mí, el sentido de responsabilidad y compromiso con mi país.

De la misma manera agradezco a mi familia, que siempre me ha dado todo su apoyo para alcanzar mis metas y quienes nunca me han faltado cuando los he necesitado.

Por último, pero con igual agradecimiento al Doctor José Abel Herrera Camacho, quien me ha encaminado en mi formación profesional del mejor modo, y que me brindó la oportunidad de trabajar con él.

Índice general

Índice de algoritmos y sistemas implementados ..... **xi**

Índice de figuras ..... **xiii**

Índice de tablas ..... **xv**

Objetivos ..... **xix**

Introducción ..... **xxiii**

1 Aspectos sobre los algoritmos de cifrado y descifrado **3**

1.1 Algoritmos de cifrado y descifrado, su clasificación 3

1.2 Propiedades matemáticas de los algoritmos de cifrado y descifrado 5

1.3 Ataques sobre los algoritmos de cifrado 6

1.4 Algoritmos de cifrado simétrico o de clave secreta 7

1.5 Algoritmos de cifrado asimétrico o de clave pública 8

2 Curvas elípticas **11**

2.1 Conjuntos 11

2.2 Operaciones binarias 13

2.3 Estructuras algebraicas de grupo, grupo abeliano y campo 13

2.4 Definición de curva elíptica 15

2.4.1 Estructura de grupo abeliano en una curva elíptica 17

2.4.2 Curvas elípticas sobre $\mathbb{Z}_p$ 19

2.4.3 Multiplicación escalar y orden de un punto en una curva elíptica 21

3 La aplicación de las curvas elípticas en los algoritmos de los sistemas criptográficos asimétricos **25**

3.1 Cota superior asintótica 26

3.1.1 Complejidad computacional de las operaciones comunes en los algoritmos. 29

3.2 Definición matemática de un sistema criptográfico asimétrico 30

3.3 El problema del logaritmo discreto 31

3.3.1 El problema del logaritmo discreto para una curva elíptica 32

3.4 Diferencias y complejidad temporal de las operaciones entre los grupos E $\mathbb{Z}_p$ y $\mathbb{Z}_p^*$ 32

3.5 Algoritmo Diffie – Hellman para el intercambio de claves 33

3.5.1 Algoritmo Diffie – Hellman para el intercambio de claves empleando curvas elípticas 34

3.6 Sistema criptográfico asimétrico ElGamal 35

3.6.1 Sistema criptográfico asimétrico ElGamal empleando curvas elípticas 37

3.7 Sistema criptográfico asimétrico Massey – Omura 39

3.7.1 Sistema criptográfico asimétrico Massey – Omura empleando curvas elípticas 40

4 Resultados **45**

4.1 Complejidad computacional y tiempo de rompimiento para el algoritmo Diffie – Hellman para el intercambio de claves 46

4.2 Complejidad computacional y tiempo de rompimiento para el sistema criptográfico asimétrico ElGamal 48

4.3 Complejidad computacional y tiempo de rompimiento para el sistema criptográfico asimétrico Massey – Omura 50

Conclusiones **55**

Apéndice A: Algoritmo extendido de Euclides e inversos modulares **59**

Apéndice B: Implementación en lenguaje Java del algoritmo extendido de Euclides con cálculo de inverso modular **61**

Apéndice C: Implementación en lenguaje Java para la suma y multiplicación escalar de puntos en una curva elíptica sobre $\mathbb{Z}_p$ **63**

Apéndice D: Implementación en lenguaje Java del algoritmo Diffie – Hellman para el intercambio de claves **69**

Apéndice E: Implementación en lenguaje Java del sistema criptográfico asimétrico ElGamal **67**

Apéndice F: Implementación en lenguaje Java del sistema criptográfico asimétrico Massey – Omura **73**

Apéndice G: Implementación en lenguaje Java del algoritmo Diffie – Hellman para el intercambio de claves empleando curvas elípticas **75**

Apéndice H: Implementación en lenguaje Java del sistema criptográfico asimétrico ElGamal empleando curvas elípticas **79**

Apéndice I: Implementación en lenguaje Java del sistema criptográfico asimétrico Massey – Omura empleando curvas elípticas **85**

Apéndice J: Glosario **91**

Apéndice K: Notación utilizada **95**

Referencias **99**

x

Índice de algoritmos y sistemas implementados

1 Algoritmo extendido de Euclides con cálculo de inverso modular 61

2 Algoritmo para la suma y multiplicación escalar de puntos en una curva elíptica sobre $\mathbb{Z}_p$ 63

3 Algoritmo Diffie – Hellman para el intercambio de claves 67

4 Sistema criptográfico asimétrico ElGamal 69

5 Sistema criptográfico asimétrico Massey – Omura 73

6 Algoritmo Diffie – Hellman para el intercambio de claves empleando curvas elípticas 75

7 Sistema criptográfico asimétrico ElGamal empleando curvas elípticas 79

8 Sistema criptográfico asimétrico Massey – Omura empleando curvas elípticas 85

Índice de figuras

Figura 2: orden de construcción de los conjuntos numéricos....12

Figura 3: curva elíptica $y^2 = x^3 - 5x$ sobre $\mathbb{R}$....16

Figura 4: curva elíptica $y^2 = x^3 + 8$ sobre $\mathbb{R}$....16

Figura 5: curva elíptica $y^2 = x^3 - 3x + 2$ sobre $\mathbb{R}$....16

Figura 6: curva elíptica $y^2 = x^3$ sobre $\mathbb{R}$....16

Figura 7: curva elíptica $E\ F\ =\ \ x, y\ \ |x, y \in F, y^2 = x^3 + Ax + B\ \cup\ \infty\ ;\ 4A^3 + 27B^2 \neq 0$....17

Figura 8: función $f\ x\ \in O\ g\ x$....27

Figura 9: Crecimiento de las complejidades computacionales obtenidas para el algoritmo Diffie – Hellman para el intercambio de claves en la versión clásica....46

Figura 10: Crecimiento de las complejidades computacionales obtenidas para el algoritmo Diffie – Hellman para el intercambio de claves en la versión para curvas elípticas....47

Figura 11: Crecimiento de las complejidades computacionales obtenidas para el sistema criptográfico asimétrico ElGamal en la versión clásica....47

Figura 12: Crecimiento de las complejidades computacionales obtenidas para el sistema criptográfico asimétrico ElGamal en la versión clásica....48

Figura 13: Crecimiento de las complejidades computacionales obtenidas para el sistema criptográfico asimétrico ElGamal en la versión para curvas elípticas....49

Figura 14: Crecimiento del tiempo de cifrado para el sistema criptográfico asimétrico ElGamal....49

Figura 15: Crecimiento del tiempo de descifrado para el sistema criptográfico asimétrico ElGamal....50

Figura 16: Crecimiento de las complejidades computacionales obtenidas para el sistema criptográfico asimétrico Massey – Omura en la versión clásica....51

Figura 17: Crecimiento de las complejidades computacionales obtenidas para el sistema criptográfico asimétrico Massey – Omura en la versión para curvas elípticas....51

Figura 18: Crecimiento del tiempo de cifrado para el sistema criptográfico asimétrico Massey – Omura....52

Figura 19: Crecimiento del tiempo de descifrado para el sistema criptográfico asimétrico Massey – Omura....52

Índice de tablas

Tabla 1: Puntos de la curva elíptica $y^2 = x^3 + 3x + 4 \bmod 7$ 20

Tabla 2: Diferencias de los elementos, operaciones, notaciones y problema del logaritmo discreto entre el grupo aditivo y el grupo multipicativo de los números enteros módulo un número primo p. 32

Tabla 3: Complejidades computacionales de algunas operaciones comunes en los algoritmos de los sistemas criptográficos asimétricos 33

Tabla 4: Complejidad computacional y tiempo de rompimiento para el algoritmo Diffie – Hellman para el intercambio de claves 46

Tabla 5: Complejidad computacional y tiempo de rompimiento para el sistema criptográfico asimétrico ElGamal 48

Tabla 6: Complejidad computacional y tiempo de rompimiento para el sistema criptográfico asimétrico Massey – Omura 50

Objetivos

Objetivos

1. Analizar y exponer las ventajas de la criptografía con curva elíptica frente a los sistemas criptográficos asimétricos actuales, mediante el estudio de los fundamentos matemáticos de su seguridad y el análisis del desempeño de sus algoritmos.

2. Implementar los algoritmos y sistemas criptográficos a estudiar en un lenguaje de programación independiente de la plataforma, para conocer su facilidad de adopción y que puedan ser apliamente aprovechados por terceras personas.

Introducción

Introducción

Cayo Julio César (Roma, Italia, 12 de julio de 100 a. C. – ibídem, 15 de marzo de 44 a. C.) fue un líder militar y político de la República Romana tardía. Fue en esta época cuando todas las grandes potencias del Mediterráneo fueron doblegadas por Roma en un corto período de tiempo; durante los siglos sucesivos Roma no volvió a tener un enemigo organizado capaz de poner en peligro su propia existencia.

Si tenía que decir algo confidencial, lo escribía usando el cifrado, esto es, cambiando el orden de las letras del alfabeto, para que ni una palabra pudiera entenderse. Si alguien quiere descifrarlo, y entender su significado, debe sustituir la cuarta letra del alfabeto, es decir, la D por la A, y así con las demás.

Suetonio, Vida de los Césares LVI[14].

De acuerdo con la vigésimo segunda edición del Diccionario de la Lengua Española, la criptografía se define como el "arte de escribir con clave secreta o de un modo enigmático". Esta definición clásica sobre la criptografía es correcta, sin embargo, no concuerda con el perfil actual de la criptografía.

Hasta mediados del siglo XX la criptografía fue un arte. El arte de diseñar métodos para ocultar información haciéndola ininteligible a intrusos, y deducir cómo la ocultaban los que existían, solamente requería destreza y habilidad personal. En sus comienzos, los usuarios de la criptografía fueron las organizaciones militares y las agencias de inteligencia, el único objetivo de la criptografía era conseguir la confidencialidad de mensajes.

La era de la criptografía moderna comienza con Claude Shannon, el padre de la *Teoría de la Información*[I]. En 1949 publicó el artículo *Communication Theory of Secrecy Systems*[15] en el Bell System Technical Journal. Sus trabajos demostraron que se podía analizar la cuantificación de la información, mediante métodos estrictamente matemáticos, lo que estableció una base teórica sólida para la criptografía.

[I] La Teoría de la Información es una rama de la teoría matemática y de las ciencias de la computación, que estudia la información y todo lo relacionado con ella.

Un *mecanismo de seguridad* es la entidad física o lógica, responsable de suministrar un servicio de seguridad. Hoy en día, todos los sistemas de cómputo ocupan mecanismos de seguridad basados en la criptografía, que además de la confidencialidad de mensajes, también se encarga de la autenticación de mensajes y de dotar de seguridad a las comunicaciones y a las entidades que se comunican. Los servicios de seguridad que brinda la criptografía son:

- *Confidencialidad:* garantiza que la información es accesible sólo para aquellos autorizados a tener acceso.
- *Integridad:* garantiza la corrección y completitud de la información.
- *No repudio:* garantiza que alguna de las entidades implicadas en una comunicación, no pueda negar haber participado en toda o parte de la misma.
- *Autenticación:* garantiza la identidad del comunicante.

No existe un único mecanismo capaz de proveer todos los servicios anteriormente listados; para suministrarlos, la criptografía hace uso de distintos mecanismos, uno de ellos es el *cifrado*, que proporciona el servicio de confidencialidad.

Un algoritmo de cifrado, es un procedimiento que utilizando cierta *clave* (la pieza de información que controla la operación del algoritmo), transforma un mensaje de tal forma que es incomprensible, o al menos, difícil de comprender, a toda entidad que no tenga la clave del algoritmo de descifrado, que se utiliza para poder hacerlo comprensible o descifrarlo. Cuando las claves de cifrado y de descifrado son iguales, se le denomina *sistema criptográfico simétrico*, y cuando son distintas, se le denomina *sistema criptográfico asimétrico*.

La mayoría de los sistemas criptográficos asimétricos actuales, fundamentan su seguridad en la resolución de un problema matemático llamado *problema del logaritmo discreto*, que debido a su gran magnitud, es casi imposible de resolver en la práctica. Ya que el estudio de estos problemas no estaba orientado para su uso criptográfico, aún no se poseen los conocimientos suficientes como para poder romperlos[II] en un tiempo razonable.

Los sistemas criptográficos que basan su seguridad en el problema del logaritmo discreto, presentan el inconveniente de que requieren mucho tiempo y recursos de procesamiento para efectuar sus operaciones. Este inconveniente los convierte en una mala elección en entornos donde los recursos son limitados y se requieren algoritmos muy eficientes que permitan ahorrar recursos de hardware.

[II] Encontrar debilidades en los sistemas y comprometer su seguridad sin el conocimiento de información secreta.

En un *sistema criptográfico de curva elíptica*, la seguridad se basa en un problema análogo al de los sistemas criptográficos asimétricos actuales, a diferencia de que en vez de operar con números enteros, opera con coordenadas de puntos de una *estructura algebraica* llamada *curva elíptica*.

Los algoritmos empleados en los sistemas criptográficos asimétricos actuales, pueden ser modificados para operar con coordenadas de puntos de una curva elíptica, brindando el mismo nivel de seguridad y con la mejoría de que requieren menor tiempo de ejecución debido al tipo de operaciones empleadas.

Pese a las ventajas que presentan los sistemas criptográficos basados en el problema del logaritmo discreto para curvas elípticas, éstos tienen el problema de que no han tenido auge por la poca difusión que tienen, debido a que su entendimiento requiere un conocimiento matemático desarrollado y a las especulaciones que existen en cuanto a su nivel de seguridad y verdadero desempeño de sus algoritmos.

Resolver el problema que ha impedido el éxito de la criptografía con curva elíptica, consiste en garantizar las ventajas que presenta sobre los sistemas criptográficos asimétricos actuales. A lo largo del presente trabajo, se estudian los fundamentos matemáticos de su seguridad y se analiza su desempeño para confirmar dichas ventajas.

El capítulo 1 muestra una definición formal de los algoritmos de cifrado y descifrado, así como los aspectos y características que deben tener los algoritmos de cifrado para que puedan ser considerados “seguros”. El capítulo 2 presenta la definición de curva elíptica, así como los aspectos matemáticos necesarios para operar con grupos de curvas elípticas.

Una vez que se tienen los antecedentes necesarios, en el capítulo 3 se trata la definición matemática de sistema criptográfico asimétrico y se estudia la teoría de la complejidad computacional para aplicarla en el análisis de dichos sistemas. Se estudian las dos versiones del problema del logaritmo discreto y su aplicación en algunos algoritmos de sistemas criptográficos asimétricos actuales, así como las modificaciones de esos algoritmos para emplearlos haciendo uso de curvas elípticas.

Finalmente, en el capítulo 4 se presentan los resultados conseguidos a lo largo de la presente obra.

Para la implementación de los algoritmos estudiados se eligió el lenguaje de programación Java, debido a que este lenguaje es independiente de la plataforma, es decir, que los programas escritos en este lenguaje se escriben una sola vez y pueden ejecutarse igualmente en cualquier dispositivo que soporte el lenguaje sin importar su

tipo de hardware. Con ello se pretende que los algoritmos que se implementaron pueden ser utilizados por terceros, principalmente en dispositivos de cómputo móvil donde el lenguaje Java ha tenido mayor éxito y que es donde se requieren sistemas criptográficos verdaderamente eficientes que eviten la pérdida de su desempeño debido a lo limitado de sus recursos, siendo la criptografía de curva elíptica quien se las puede brindar.

Capítulo 1

Aspectos sobre los algoritmos de cifrado y descifrado

Capítulo 1

Aspectos sobre los algoritmos de cifrado y descifrado

1.1 Algoritmos de cifrado y descifrado, su clasificación

Un *algoritmo* es un procedimiento computacional bien definido, que toma una variable de entrada y termina con una salida. Un *algoritmo de cifrado* es un algoritmo para ocultar por transformaciones, un texto ordinario, llamado *texto plano*, en un *texto cifrado* de tal forma que sea incomprensible, o al menos, difícil de comprender. Este proceso es llamado cifrado de texto plano en texto cifrado, el proceso inverso es llamado descifrado (figura 1).

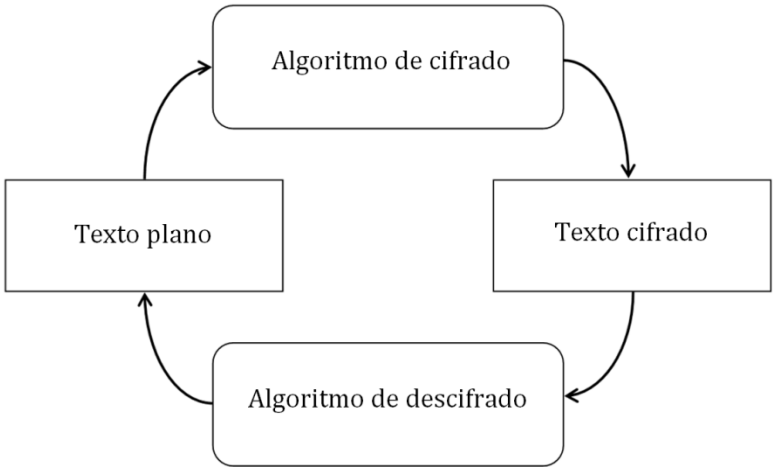

Figura 1: proceso de cifrado y descifrado de un texto.

Los algoritmos de cifrado se dividen en dos categorías: *algoritmos de cifrado de sustitución* y *algoritmos de cifrado de transposición*. Los algoritmos de cifrado de sustitución reemplazan letras o grandes bloques del texto original con sustitutos, usualmente de la misma longitud. En un algoritmo de cifrado de sustitución simple, el mismo alfabeto es usado para el texto plano y el texto cifrado, y una permutación fijada en este alfabeto da la regla de sustitución.

Como ejemplo, supóngase que las letras del alfabeto están ordenadas en un círculo (empezando con A y siguiendo hasta la Z) y el cifrado del mensaje se realiza reemplazando cada letra del texto plano, por la quinta letra que le sigue en el círculo. Así el mensaje $SEGURO$ puede ser cifrado como $XJLZW$. El descifrado se realiza sustituyendo cada letra del texto cifrado, por la quinta letra antes de ella en el alfabeto en círculo. Este algoritmo de cifrado en el cual el alfabeto se rota, es llamado *algoritmo de cifrado César* por que Julio César lo utilizó.

En un algoritmo de cifrado de transposición, el texto cifrado utiliza las mismas letras y las mismas frecuencias que las del el texto plano, solo que estas son reordenadas. Un ejemplo simple de algoritmo de cifrado de transposición usa una matriz. El texto plano y el texto cifrado, son partidos en bloques con longitud igual al número de entradas en la matriz. Un mensaje se cifra escribiendo cada bloque de texto plano en la matriz por renglones, la salida de un bloque de texto cifrado con la matriz, se lee por columnas hacia atrás.

Por ejemplo, supóngase que se usa una matriz de $2\,x\,3$ para cifrar el mensaje $SEGURO$, el cual tiene solo un bloque. Se forma la matriz:

$$\begin{matrix} S & E & G \\ U & R & O \end{matrix}$$

y se lee el texto cifrado $GOERSU$. El descifrado se realiza escribiendo cada bloque de texto cifrado en la matriz por columnas hacia atrás, y el bloque de texto plano se lee por renglones.

Los *algoritmos de cifrado producto* son creados por la composición de varios algoritmos de cifrado, cuyos tipos alternan entre sustitución y trasposición. Cada algoritmo de cifrado de sustitución y trasposición tiene ciertas debilidades, las cuales pueden ser superadas por composición de ellos en forma alternada. Por ejemplo, los dos algoritmos de cifrado anteriores se pueden componer, usando el algoritmo de cifrado César primero. El texto plano $SEGURO$ es primero cambiado a $GOERSU$. Este se escribe en la matriz:

$$\begin{matrix} G & O & E \\ R & S & U \end{matrix}$$

y el texto cifrado es $EUOSGR$.

Tanto el algoritmo de cifrado como el algoritmo de descifrado son controlados por *claves*, es decir, la pieza de información que controla la operación del algoritmo. La clave para un algoritmo de cifrado de trasposición es la permutación fijada en las letras de un bloque. La clave para un algoritmo de cifrado de sustitución simple, es la permutación fijada en el alfabeto. En el caso simple del algoritmo de cifrado César, la clave es la cantidad de desplazamiento del alfabeto.

1.2 Propiedades matemáticas de los algoritmos de cifrado y descifrado

Supóngase qué E y D son algoritmos prototipo de cifrado y descifrado, respectivamente, antes de que la clave sea especificada. Cuando la clave es K, al especificar K en E y D, se obtienen las funciones de cifrado y descifrado, a menudo escritas como E_K y D_K, respectivamente. Así, $C = E_K\ M$ significa que C es el texto cifrado obtenido cuando el texto plano M es cifrado con la clave K. De la misma manera, $M = D_K\ C$ significa que C fue descifrado utilizando la clave K para obtener M.

Una propiedad fundamental compartida por todas las funciones de cifrado y descifrado, es que $D_K\ E_K\ M\ = M$ para toda M. Esta ecuación dice que si se cifra un mensaje M con la clave K, entonces se puede recuperar M utilizando la misma clave K para descifrarlo.

En general, las funciones E_K y D_K deben ser rápidas para todas las claves K, y la seguridad del cifrado solo debe depender de la secrecía de las claves y no de la secrecía de los métodos E y D. Estos requerimientos son importantes, pues un espía podría conocer los métodos E y D que siempre son fijos, mientras que la clave K puede ser cambiada frecuentemente; en algunas funciones de cifrado, todas las claves con la misma longitud tienen la misma seguridad, pero en otros, las claves deben ser elegidas de acuerdo a propiedades especiales para que puedan funcionar o ser seguras.

1.3 Ataques sobre los algoritmos de cifrado

El *criptoanálisis* es el estudio de los *ataques* sobre los algoritmos de cifrado. Un ataque sobre un algoritmo de cifrado, es un método por el cual un individuo intenta encontrar debilidades en dicho algoritmo y comprometer su seguridad sin el conocimiento de información secreta. Los métodos de ataque pueden ser clasificados en muchos tipos generales de acuerdo al tipo de información que es conocida o desconocida para el *criptoanalista*[III].

En un ataque de *solo texto cifrado*, solo se conoce el texto cifrado, aunque a menudo el lenguaje del texto plano y el tipo de algoritmo de cifrado también son conocidos. El objetivo del criptoanalista es encontrar el texto plano y la clave. Este es el tipo de ataque más difícil. Algunas veces el criptoanalista solo tiene una cadena de bits con que trabajar.

En un ataque de *texto plano conocido*, el criptoanalista tiene algún texto cifrado y su correspondiente texto plano. Por ejemplo, se puede saber que todos los mensajes que un usuario A envía a un usuario B tienen la misma cabecera. En este caso, la primer parte de cada texto cifrado puede ser descifrado porque siempre es la misma. El objetivo es encontrar la clave con la que otro texto cifrado puede ser descifrado.

En un ataque de *texto plano escogido*, el criptoanalista puede especificar algún texto plano a cifrar, tal vez un texto sin significado, y de alguna manera aprender como es el correspondiente texto cifrado. Esta hazaña puede ser realizada al engañar al operador de la máquina de cifrado y darle un mensaje a cifrar o capturando un fragmento cifrado con una clave ilegible guardada en él. El objetivo es encontrar la clave.

Un buen algoritmo de cifrado debería resistir todos estos tipos de ataques. Específicamente, para un criptoanalista debería ser computacionalmente irrealizable, efectuar alguna de las siguientes acciones, sin importar que tanto texto cifrado tenga:

1. Encontrar M dado C.
2. Encontrar D_K dado C o C y el correspondiente M.
3. Construir C de modo que D_K C es cualquier mensaje útil.
4. Encontrar E_K dado C o C y el correspondiente M.

Los dos primeros requerimientos aseguran la secrecía de la función de cifrado y los mensajes cifrados con él. Los requerimientos 1 y 2 dicen que un ataque de solo texto

[III] La persona que se dedica al criptoanálisis.

cifrado debería de ser difícil. El requerimiento 2 dice que un ataque de solo texto conocido debería ser difícil.

Los últimos dos requerimientos aseguran la autenticidad de mensajes cifrados con el algoritmo de cifrado. El requerimiento 3 dice que un atacante no puede crear texto cifrado con el que podría descifrar un texto plano útil, aunque este texto plano pueda ser extraño, e incluso, desconocido para el atacante. El requerimiento 4 dice que un atacante no puede descubrir la función de cifrado, utilizándola para cifrar un mensaje falso (como una transferencia bancaria), y tener el recibo de aceptado como auténtico. Ambos dicen que si un atacante activo reemplaza un texto cifrado con otro, el cambio es casi seguro que se detecte.

1.4 Algoritmos de cifrado simétrico o de clave secreta

En una comunicación, las dos partes que intervienen deben ponerse de acuerdo de antemano sobre la clave a usar. Una vez que ambas partes tienen acceso a esta clave, el remitente cifra un texto usando la clave, lo envía al destinatario y éste lo descifra con la misma clave.

Cuando en un sistema criptográfico la misma clave es usada por los algoritmos de cifrado y descifrado, el cifrado es llamado *cifrado de una clave*, *cifrado simétrico*, *cifrado convencional* o *cifrado de clave secreta*.

Como se mencionó, la seguridad de un cifrado solo debe depender de la secrecía de las claves y no de la secrecía de los métodos E y D, en otras palabras, no debería ser de ninguna ayuda para un criptoanalista conocer los algoritmos de cifrado y descifrado que se están empleando.

El principal problema con los sistemas criptográficos que emplean algoritmos de cifrado simétrico no está ligado a su seguridad, sino al intercambio de claves. Una vez que el remitente y el destinatario han intercambiado las claves pueden usarlas para comunicarse con seguridad, sin embargo, estos sistemas presentan el problema de la carencia de un método para el intercambio de claves de manera segura.

Otro problema es el número de claves que se necesitan. Si se tiene un x número de usuarios que necesitan comunicarse entre sí, se necesitan $x/2$ claves para cada pareja de usuarios que tengan que comunicarse de manera privada.

1.5 Algoritmos de cifrado asimétrico o de clave pública

Hasta la década de 1970, los criptógrafos asumían que si alguien conocía una función de cifrado E_K, incluyendo su clave K, entonces ese alguien podría fácilmente deducir la función correspondiente de descifrado D_K. Todos los algoritmos de cifrado inventados hasta ese entonces fueron de este tipo y se asumía que este inconveniente no podía ser eliminado.

En 1976 Whitfield Diffie y Martin Hellman[16] propusieron un nuevo tipo de algoritmo de cifrado llamado *cifrado asimétrico*, para el cual este inconveniente no influye. Cada usuario del nuevo algoritmo debería tener una función de cifrado la cual podría hacer pública, y una función de descifrado, la cual debería mantener en secreto.

En los actuales sistemas criptográficos que emplean algoritmos de cifrado asimétrico, ambos métodos, E y D, son públicos y los mismos para cada usuario. Sin embargo, las dos funciones para un usuario tienen diferentes claves, la clave de cifrado es pública y la clave de descifrado es secreta. Computacionalmente no es factible deducir la clave de descifrado a partir de la clave de cifrado y viceversa.

Cuando en un sistema criptográfico la clave empleada por el algoritmo para cifrar es diferente a la empleada por el algoritmo para descifrar y ninguna de ellas puede ser fácilmente deducible a partir de la otra, el cifrado es llamado *cifrado de dos claves*, *cifrado asimétrico* o *cifrado de clave pública*.

Los sistemas criptográficos que emplean algoritmos de cifrado asimétrico, eliminan por completo el problema del intercambio de claves de manera segura. Con las claves públicas no es necesario que el remitente y el destinatario se pongan de acuerdo en la clave a emplear, todo lo que se requiere es que antes de iniciar la comunicación secreta el remitente consiga una copia de la clave pública del destinatario.

Estos sistemas de cifrado también eliminan el problema del número de claves, pues la misma clave pública puede ser usada por cualquiera que desee comunicarse con su propietario, es decir, sólo se necesitarán n pares de claves por cada n usuarios que deseen comunicarse entre sí.

En el capítulo 3 se estudian los fundamentos matemáticos empleados por los algoritmos para el intercambio de claves de manera segura y se trata la definición matemática de los sistemas criptográficos asimétricos.

Capítulo 2

Curvas elípticas

representar conjuntos, y minúsculas para representar los objetos pertenecientes a los mismos.

Si $X = \{a, b, c, d\}$, entonces a, b, c y d, se llaman *miembros* o *elementos* del conjunto X. La notación $a \in X$ se lee "a es un elemento que pertenece al conjunto X". Para denotar que un objeto e no pertenece al conjunto X, se escribe $e \notin X$ y se lee "e es un elemento que no pertenece al conjunto X".

La segunda manera de describir un conjunto consiste en proporcionar la regla que identifica a sus elementos. Dicha regla se escribe entre llaves: $N = \{\text{todos los números naturales}\}$.

Una *variable* es una literal que adquiere varios valores en un problema dado. Para encontrar un miembro genérico de un conjunto de números, se emplea una variable como x, y, z, etc. El conjunto X cuyos elementos cumplen una propiedad P se denota por: $X = \{x | x \text{ tiene la propiedad } P\}$, lo cual se lee "$X$ es el conjunto de elementos x, tales que x tienen la propiedad P".

Los conjuntos numéricos son agrupaciones de números que guardan una serie de propiedades estructurales. Sus características más importantes son:

- No son conjuntos finitos.
- Todos los conjuntos numéricos se construyen desde una estructura más simple hasta otra más compleja.
- Todos los conjuntos numéricos son a su vez, subconjuntos del conjunto $\mathbb{C}$ de los números complejos.

El orden de construcción de los conjuntos numéricos es el siguiente (figura 2):

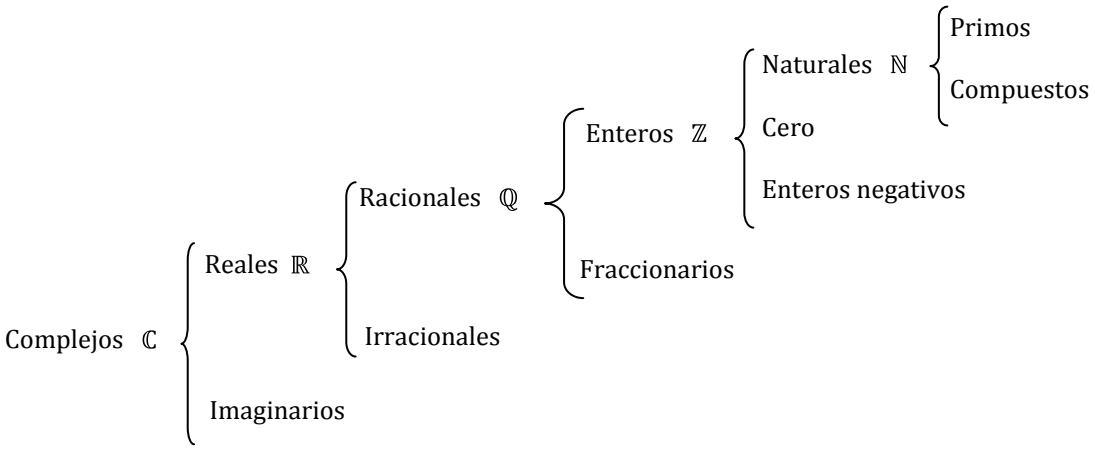

Figura 2: orden de construcción de los conjuntos numéricos.

Capítulo 2

Curvas elípticas

Una curva elíptica es una curva plana definida mediante una ecuación de tercer grado, en la que se puede definir una operación binaria para el conjunto de sus puntos y que constituye una estructura de grupo abeliano. El término curva elíptica es un nombre inapropiado ya que las curvas elípticas no son elipses; el término proviene del hecho de que las curvas elípticas hicieron su aparición inicial durante los primeros intentos para calcular la longitud de arco de una elipse.

Los grupos de curvas elípticas tienen muchas propiedades que los hacen útiles para la criptografía, estos grupos pueden ser usados directamente en algoritmos criptográficos.

2.1 Conjuntos

Un conjunto es una agrupación de objetos simples en un todo. Un conjunto es un medio por el cual se puede hablar de colecciones de objetos de manera abstracta.

Los números 1, 2, 3, etc., constituyen un conjunto que se llama *conjunto de los números naturales*, denotado por N. No se supone ninguna propiedad uniforme de los objetos que forman un conjunto, salvo que están agrupados para constituirlo.

Existen dos maneras de describir un conjunto, la primera consiste en hacer una lista de los objetos que lo componen y separarlos con coma, dicha lista se escribe entre llaves { }. Por ejemplo, $N = 1, 2, 3 \ldots$. Se acostumbra emplear letras mayúsculas para

2.2 Operaciones binarias

Una operación binaria $*$ definida en un conjunto S no vacío, es una función de $S\ x\ S$ en S. La imagen del par ordenado a, b bajo la operación $*$ se representa con $a * b$. Una operación binaria $*$ definida en un conjunto S asigna siempre como resultado un elemento de S; es decir que:

$$\forall\ a, b \in S: \quad a * b \in S$$

Esta situación se define diciendo que el conjunto S es *cerrado* respecto a la operación $*$. Algunos ejemplos de operaciones binarias conocidas son:

- La adición y la multiplicación en el conjunto de los números naturales.
- La sustracción en el conjunto de los números enteros.
- La adición y la sustracción de polinomios.
- La unión y la intersección de conjuntos, etc.

Cuando en un conjunto S se han definido una o varias operaciones binarias $*$, Δ, …, se acostumbra denotarlo como $S, *, \Delta, \ldots,$ y se dice que las operaciones $*$, Δ, …, determinan una *estructura algebraica* en S, y que $S, *, \Delta, \ldots,$ es un *sistema algebraico*.

2.3 Estructuras algebraicas de grupo, grupo abeliano y campo

Definiciones:

a) Elementos idénticos

Sea $*$ una operación binaria definida en un conjunto S:

1. Un elemento $e \in S$ es un idéntico izquierdo para $*$ si:

$$e * a = a, \quad \forall\ a \in S$$

2. Un elemento $e \in S$ es un idéntico derecho para $*$ si:

$$a * e = a, \quad \forall\ a \in S$$

3. Un elemento $e \in S$ es un idéntico para $*$ si es idéntico izquierdo e idéntico derecho.

b) Elementos inversos

Sea $*$ una operación binaria definida en un conjunto S:

1. Sea e un idéntico izquierdo para $*$. Un elemento $a \in S$ es un inverso izquierdo del elemento $a \in S$ para $*$ si:

$$a * a = e$$

2. Sea e un idéntico derecho para $*$. Un elemento $a \in S$ es un inverso derecho del elemento $a \in S$ para $*$ si:

$$a * a = e$$

3. Sea e un idéntico para $*$. Un elemento $a \in S$ es un inverso del elemento $a \in S$ para $*$ si:

$$a * a = e \quad \text{y} \quad a * a = e$$

c) Asociatividad

Sea $*$ una operación binaria definida en un conjunto S:

Se dice que * es asociativa si:

$$\forall\, a, b, c \in S: \quad a * \ b * c \ = \ a * b \ * c$$

d) Conmutatividad

Sea $*$ una operación binaria definida en un conjunto S:

Se dice que * es conmutativa si:

$$\forall\, a, b \in S: \quad a * b = b * a$$

e) Propiedades distributivas

Cuando dos operaciones $+$ y $\cdot$, definidas en un conjunto S, son tales que:

$$\forall\, a, b, c \in S: \quad a \cdot \ b + c \ = \ a \cdot b \ + \ a \cdot c$$

se dice que la operación $\cdot$ es distributiva por la izquierda sobre la operación $+$, y cuando son tales que:

$$\forall\, a, b, c \in S: \quad b + c \ \cdot a = \ b \cdot a \ + \ c \cdot a$$

se dice que · es distributiva por la derecha sobre +.

f) Estructura de grupo

Sea G un conjunto no vacío y sea $*$ una operación definida en G. El sistema algebraico $\langle G, * \rangle$ tiene estructura de grupo si:

1. $\forall\, a, b, c \in G$: $\quad a * (b * c) = (a * b) * c$
2. $\exists\, e \in G$ tal que $\quad e * a = a, \quad \forall\, a \in G$
3. $\forall\, a \in G \quad \exists\, a \in G$ tal que $\quad a * a = e, \quad \forall\, a \in G$

g) Estructura de grupo abeliano

Un grupo $\langle G, * \rangle$ se dice que es abeliano si:

$$\forall\, a, b \in G: \quad a * b = b * a$$

h) Estructura de campo

Sea F un conjunto de por lo menos dos elementos, y sean + y · dos operaciones binarias definidas en F. El sistema algebraico $\langle F, +, \cdot \rangle$ es un campo si:

1. $\langle F, + \rangle$ es un grupo abeliano, cuyo elemento idéntico denotamos con O.
2. $\langle F - O, \cdot \rangle$ es un grupo abeliano.
3. · es distributiva por la izquierda y por la derecha sobre +.

2.4 Definición de curva elíptica

Dado un campo F, se llama *curva elíptica sobre F,* a la curva plana sobre F definida por la ecuación:

$$y^2 = x^3 + Ax + B$$

donde x, y, A y B son números reales, números racionales o números enteros que pertenecen a F.

El conjunto:

$$E(F) = \{(x, y) \mid x, y \in F, y^2 = x^3 + Ax + B\} \cup \{\infty\}$$

forma un grupo abeliano; el punto ∞ no es un punto sobre la gráfica $y^2 = x^3 + Ax + B$ (figuras 3 y 4), es un punto llamado *punto en el infinito*, que es el elemento identidad del grupo de la curva elíptica.

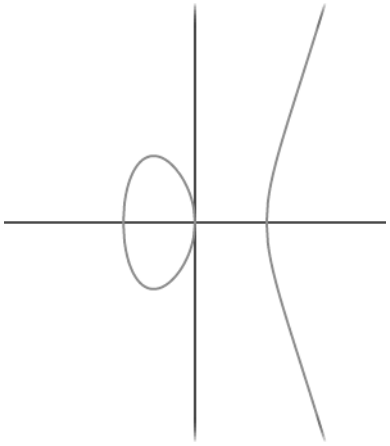

Figura 3: curva elíptica $y^2 = x^3 - 5x$ sobre $\mathbb{R}$.

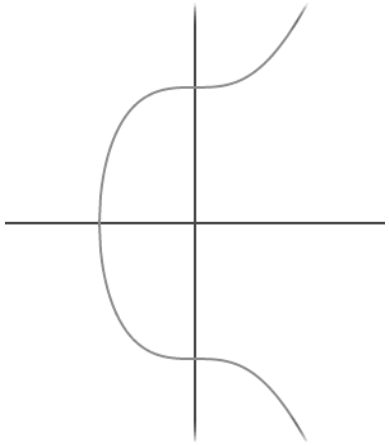

Figura 4: curva elíptica $y^2 = x^3 + 8$ sobre $\mathbb{R}$.

Si las raíces de este polinomio de tercer grado son r_1, r_2, r_3; entonces el discriminante del polinomio es:

$$r_1 - r_2 \quad r_1 - r_3 \quad r_2 - r_3 \quad ^2 = - \ 4A^3 + 27B^2$$

Cuando $4A^3 + 27B^2 \neq 0$, esta condición asegura que la ecuación $y^2 = x^3 + Ax + B$ no tenga raíces repetidas y permite excluir las curvas elípticas que tienen un *punto doble* (figura 5) o un *pico* (figura 6).

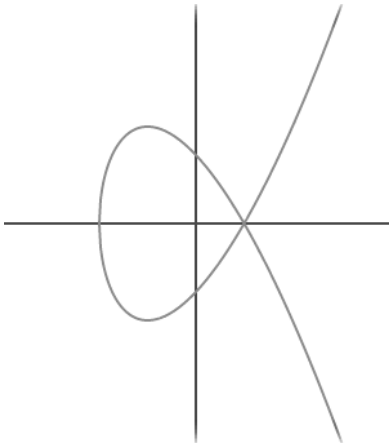

Figura 5: curva elíptica $y^2 = x^3 - 3x + 2$ sobre $\mathbb{R}$. Presenta un punto doble cuando $y = 0$.

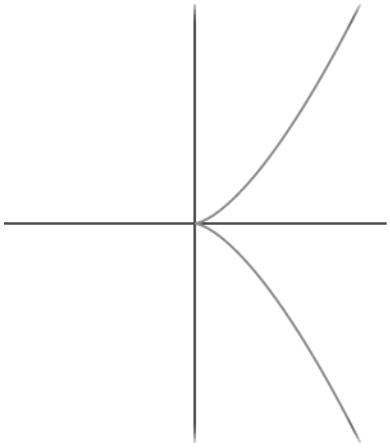

Figura 6: curva elíptica $y^2 = x^3$ sobre $\mathbb{R}$. Presenta un pico cuando $y = 0$.

2.4.1 Estructura de grupo abeliano en una curva elíptica

Sean $P_1 = (x_1, y_1)$ y $P_2 = (x_2, y_2)$ dos puntos sobre una curva elíptica E con discriminante $4A^3 + 27B^2 \neq 0$ (figura 7), el segmento P_1P_2 corta a E en tres puntos (porque la ecuación que define a E es de tercer grado), el nuevo punto P_3 se obtiene como a continuación se indica.

Se dibuja una línea L a través de P_1 y P_2, L interseca a E en un tercer punto P_3'. Al reflejar P_3' a través del eje x (cambia el signo de la coordenada y) se obtiene P_3 y se define que: $P_1 + P_2 = P_3$; entonces se puede introducir una operación binaria llamada adición y denotada por "+".

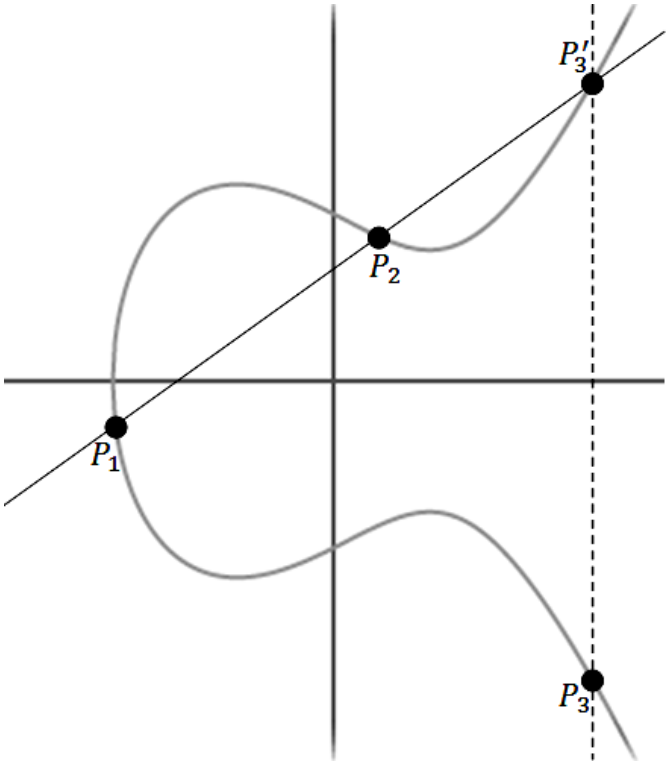

Figura 7: curva elíptica $E\ F\ =\ \ x, y\ \mid x, y \in F, y^2 = x^3 + Ax + B\ \cup\ \infty\ ;\ 4A^3 + 27B^2 \neq 0$

El ejemplo anterior se refiere a la adición de puntos sobre una curva elíptica, que no es lo mismo que la adición de coordenadas de los puntos.

Se asume que $P_1 \neq P_2$ y que tampoco es el punto en el infinito ∞. La pendiente de la línea L dibujada a través de P_1 y P_2 es:

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}$$

también se asume que $x_1 \neq x_2$, entonces la ecuación de L es:

$$y = \lambda\ x - x_1\ + y_1$$

para obtener la intersección con E, se sustituye para obtener:

$$\lambda\ x - x_1\ + y_1\ ^2 = x^3 + Ax + B$$

esta puede reordenarse de la forma:

$$0 = x^3 - \lambda^2 x^2 + \cdots$$

Las tres raíces de este polinomio corresponden a los tres puntos de intersección de L con E. Ya se conocen dos de las raíces de esta ecuación, llamadas x_1 y x_2, que pertenecen a los puntos P_1 y P_2 respectivamente, y que ambos están sobre L y E. Es posible factorizar el polinomio para obtener el tercer valor de x, pero existe un método más fácil. Si se tiene un polinomio de tercer grado: $x^3 + ax^2 + bx + c$ con raíces r, s, t, entonces:

$$x^3 + ax^2 + bx + c = \ x - r\ \ x - s\ \ x - t\ = x^3 - \ r + s + t\ x^2 + \cdots$$

Por lo tanto, $r + s + t = -a$. Si se conocen las dos raíces r, s, entonces es posible hallar la tercera como $t = -a - r - s$. En este caso, se obtiene: $x = \lambda^2 - x_1 - x_2$ y $y = \lambda\ x - x_1\ - y_1$. Ahora, reflejando a través del eje x para obtener el punto P_3:

$$x_3 = \lambda^2 - x_1 - x_2, \quad y_3 = \lambda\ x_1 - x_3\ - y_1$$

Considerando el caso donde $x_1 = x_2$, pero $y_1 \neq y_2$; la línea a través de P_1 y P_2 es una línea vertical, por lo tanto interseca a E en ∞. Reflejando ∞ a través del eje x produce el mismo punto ∞ (esto es porqué se puso ∞ en la cima y en el fondo del eje y). Por lo tanto, en este caso $P_1 + P_2 = \infty$.

Ahora considérese el caso donde $P_1 = P_2 = (x_1, y_1)$. Cuando dos puntos en una curva están muy cercanos el uno al otro, la línea a través de ellos se aproxima a la línea tangente. Por lo tanto, cuando los dos puntos coinciden, se toma la línea L a través de ellos para que sea la línea tangente. La derivación implícita permite encontrar la pendiente λ de L:

$$2y\frac{dy}{dx} = 3x^2 + A, \quad \text{así que} \quad \lambda = \frac{dy}{dx} = \frac{3x_1^2}{2y_1}$$

Si $y_1 = 0$ entonces la línea es vertical, ya se dijo que $P_1 + P_2 = \infty$ (si $y_1 = 0$, entonces el numerador $3x^2 + A \neq 0$). Por lo tanto, se asume que $y_1 \neq 0$. La ecuación de L (como antes) es:

$$y = \lambda\ x - x_1\ + y_1$$

Así que se obtiene la ecuación cúbica:

$$0 = x^3 - \lambda^2 x^2 + \cdots$$

Esta vez, se sabe que solo tiene una raíz, llamada x_1, pero es una raíz doble ya que L es tangente a E en P_1. Por lo tanto, procediendo como antes, se obtiene:

$$x_3 = \lambda^2 - 2x_1, \qquad y_3 = \lambda\ x_1 - x_3\ - y_1$$

Finalmente, supóngase que $P_2 = \infty$. La línea a través de P_1 e ∞ es una línea vertical que interseca E en el punto P_1' que es la reflexión de P_1 a través del eje x. Cuando se refleja P_1' a través del eje x para obtener $P_3 = P_1 + P_2$ se regresa al punto P_1. Por lo tanto:

$$P_1 + \infty = P_1$$

Para todos los puntos P_1 sobre E. Por supuesto, esto se extiende para incluir $\infty + \infty = \infty$.

Cuando P_1 y P_2 tienen coordenadas en un campo L que contiene A y B, entonces $P_1 + P_2$ también tiene coordenadas en L. Por lo tanto $E\ L$ es cerrada bajo la suma cerrada de punto

La adición de puntos sobre una curva elíptica E satisface las siguientes propiedades:

1. *Conmutatividad:* $P_1 + P_2 = P_2 + P_1$ para todo P_1, P_2 sobre E.
2. *Existencia de elemento idéntico:* $P + \infty = P$ para todos los puntos P sobre E.
3. *Existencia de elementos inversos:* Dado un punto P sobre E, existe un punto P' sobre E con $P + P' = \infty$. Este punto P' usualmente se denota – P.
4. *Asociatividad:* $P_1 + P_2\ + P_3 = P_1 +\ P_2 + P_3$ para todo P_1, P_2, P_3 sobre E.

Se demuestra que esto convierte a la curva E en un grupo abeliano aditivo con ∞ como el elemento identidad.

2.4.2 Curvas elípticas sobre $\mathbb{Z}_p$

Una curva elíptica E sobre $\mathbb{Z}_p$, tiene la ecuación de la forma:

$$y^2 = x^3 + Ax + B \bmod p$$

se define el conjunto:

$$E\ \mathbb{Z}_p\ =\ \ x,y\ \ |x,y\ \in\ \mathbb{Z}_p, y^2 = x^3 + Ax + B \bmod p\ \ \cup\ \ \infty$$

que forma un grupo abeliano y $A, B, x, y\ \in\ \mathbb{Z}_p$. La condición de discriminante se convierte en $4A^3 + 27B^2 \neq 0 \bmod p$.

Es importante saber que un elemento $y\ \in\ \mathbb{Z}_p$ es un *residuo cuadrático módulo* p si existe un $y\ \in\ \mathbb{Z}_p$ tal que $x^2 = y \bmod p$; decimos que x es una raíz cuadrada de y en este caso. Además, cuando $p > 2$ es primo, todo residuo cuadrático modulo p tiene exactamente dos raíces cuadradas.

Por supuesto, la gráfica no es una curva en el plano; solo es un conjunto de pares módulo p. Por ejemplo, sea la curva elíptica:

$$E\ \mathbb{Z}_7\ =\ \ x,y\ \ |x,y\ \in\ \mathbb{Z}_7, y^2 = x^3 + 3x + 4 \bmod 7\ \ \cup\ \ \infty$$

x	$x^3 + 3x + 4 \bmod 7$	y	$y^2 \bmod 7$
0	4	2	4
0	4	5	4
1	1	1	1
1	1	6	1
2	4	2	4
2	4	5	4
3	5	–	$\nexists\ y \in \mathbb{Z}_7 \mid y^2 \bmod 7 = 5$
4	3	–	$\nexists\ y \in \mathbb{Z}_7 \mid y^2 \bmod 7 = 3$
5	4	2	4
5	4	5	4
6	0	0	0

Tabla 1: Puntos de la curva elíptica $y^2 = x^3 + 3x + 4 \bmod 7$

La curva elíptica E tiene diez puntos incluyendo el punto en el infinito[IV]: $0,2\ ,\ \ 0,5\ ,\ \ 1,1\ ,\ \ 1,6\ ,\ \ 2,2\ ,\ \ 2,5\ ,\ \ 5,2\ ,\ \ 5,5\ ,\ \ 6,0\ , \infty$.

La operación suma de puntos para una curva elíptica $E\ \mathbb{Z}_p$ se resume así:

1. $\infty + \infty = \infty$.
2. $P_1 + \infty = \infty + P_1 = P_1\ \forall\ P_1 \in E\ \mathbb{Z}_p\ .$
3. Si $P_1 = (x_1, y_1) \in E\ \mathbb{Z}_p$, entonces $x_1, y_1\ +\ x_1, -y_1\ = \infty$ (el punto $x_1, -y_1$ es un punto sobre $E\ \mathbb{Z}_p$ llamado *negativo de* P_1 y es denotado como $-P_1$.
4. Sea $P_1 = (x_1, y_1) \in E\ \mathbb{Z}_p$ y $P_2 = (x_2, y_2) \in E\ \mathbb{Z}_p$, donde $x_1 \neq x_2$. Entonces $P_1 + P_2 = P_3 =\ x_3, y_3$ es:

[IV] El orden de una curva elíptica E definida sobre un campo F, es el número de puntos sobre la curva elíptica E incluyendo el punto en el infinito (∞).

$$x_3 = \lambda^2 - x_1 - x_2 \bmod p$$

$$y_3 = \lambda\ x_1 - x_3\ - y_1 \bmod p, \text{ donde}$$

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} \bmod p$$

5. Sean $P_1 = P_2 = (x_1, y_1) \in E\ \mathbb{Z}_p$, donde $y_1 \neq 0$. Entonces $P_1 + P_2 = P_3 = x_3, y_3$ es :

$$x_3 = \lambda^2 - 2x_1 \bmod p$$

$$y_3 = \lambda\ x_1 - x_3\ - y_1 \bmod p, \text{ donde}$$

$$\lambda = \frac{3x_1^2}{2y_1} \bmod p$$

Todas las operaciones anteriores se realizan haciendo uso de la aritmética modular[V]; en el caso de la división en aritmética modular, cuando el módulo (el residuo de la división) es cero, la operación procede de manera normal, pero si es distinto de cero, la división se define como multiplicar el numerador por el *inverso modular*[VI] del denominador.

2.4.3 Multiplicación escalar y orden de un punto en una curva elíptica

Si k es un entero positivo, y P un punto sobre una curva elíptica E; entonces $kP = \ _{i=1}^{k} P$ denota el punto obtenido de sumar k copias del punto P a si mismo. El proceso de calcular kP a partir de P y k es llamado *multiplicación escalar*.

El *orden de un punto* P es el entero positivo k más pequeño tal que $kP = \infty$ (el punto en el infinito).

[V] En el apéndice C se encuentra una implementación en lenguaje Java para la suma y multiplicación escalar de puntos en una curva elíptica sobre $\mathbb{Z}_p$.
[VI] Ver el apéndice A.

Capítulo 3

La aplicación de las curvas elípticas en los algoritmos de los sistemas criptográficos asimétricos

Capítulo 3

La aplicación de las curvas elípticas en los algoritmos de los sistemas criptográficos asimétricos

Los grupos de curvas elípticas tienen muchas propiedades que los hacen útiles para la criptografía, estos grupos pueden ser usados directamente en algoritmos criptográficos como los empleados en los sistemas criptográficos asimétricos actuales. En estas aplicaciones, el problema del logaritmo discreto es igual de fuerte para grupos de curvas elípticas, que para los enteros módulo p, con la ventaja de que permiten algoritmos más rápidos.

El objetivo de un problema computacional siempre es encontrar el algoritmo más eficiente (por ejemplo, el más rápido) para resolverlo. La *teoría de la complejidad computacional* es una rama de la teoría de la computación, que estudia de manera teórica, la complejidad inherente a la resolución de un problema computable. Los recursos comúnmente estudiados son el *tiempo,* mediante una aproximación al número y tipo de operaciones[7] utilizadas por un algoritmo para resolver un problema, y el *espacio*, mediante una aproximación a la cantidad de memoria utilizada para resolver el problema.

La *complejidad temporal* de un problema, es el número de operaciones que toma resolver el problema a partir del tamaño de la entrada[8]. Si se toma una instancia del

[7] El tiempo de ejecución de un algoritmo sobre una entrada en particular, es el número de operaciones primitivas o "pasos" ejecutados.

[8] El tamaño de entrada n, es el número total de bits necesarios para representar la entrada en notación binaria ordinaria. El número de bits en representación binaria ordinaria de un entero positivo p, es $1 + \log p$ bits.

problema con una entrada de longitud n, que puede resolverse en n^2 pasos, se dice que ese problema tiene una complejidad en tiempo n^2.

Actualmente las computadoras pueden resolver problemas mediante algoritmos que tienen una *complejidad polinomial*; es decir, cuando el tiempo de ejecución de un algoritmo, es menor que un cierto valor calculado, usando una fórmula polinomial a partir del número de variables implicadas. Dentro de los tiempos polinomiales se distinguen los logarítmicos (log n), lineales (n), cuadráticos (n^2), cúbicos (n^3), etc.

Los problemas que tienen una complejidad factorial ($n!$), exponencial (a^n, donde a es una constante y $n > 1$) o combinatoria, en la actualidad no tienen una solución práctica, es decir, no pueden ser resueltos en un tiempo razonable.

El tiempo para resolver un problema no solo depende del número de operaciones que realiza, también depende de otros factores como el hardware, software y el lenguaje de programación empleado para implementar un algoritmo para resolverlo. Cuando se tiene un algoritmo para resolver un problema con complejidad en tiempo n, es posible calcular el tiempo que se requerirá para resolverlo en una plataforma específica, multiplicando el tiempo anteriormente calculado por una constante.

La cota superior asintótica, permite estimar el número de operaciones que realiza un algoritmo a medida que la longitud de su entrada crece, y con ello determinar si es práctico utilizar dicho algoritmo. También permite comparar distintos algoritmos y decidir cuál es el más eficiente para la solución de un determinado problema.

3.1 Cota superior asintótica

Una cota superior asintótica, es una función que sirve de cota superior de otra función, cuando el argumento tiende a infinito. Usualmente se utiliza la notación $\mathcal{O}$ de Landau, coloquialmente llamada "notación de la o grande", para referirse a las funciones acotadas superiormente por otra función.

Si f y g son dos funciones en el conjunto de los números reales, una función $f\ x\ \in \mathcal{O}\ g\ x$ si existen dos constantes C y k tales que $f\ x\ \leq C\ g\ x$ siempre que $x > k$, dichas constantes C y k se llaman *testigos* de la relación $f\ x\ \in \mathcal{O}\ g\ x$. La relación $f\ x\ \in \mathcal{O}\ g\ x$ a menudo se lee "$f\ x$ es o grande de $g\ x$ " (figura 8).

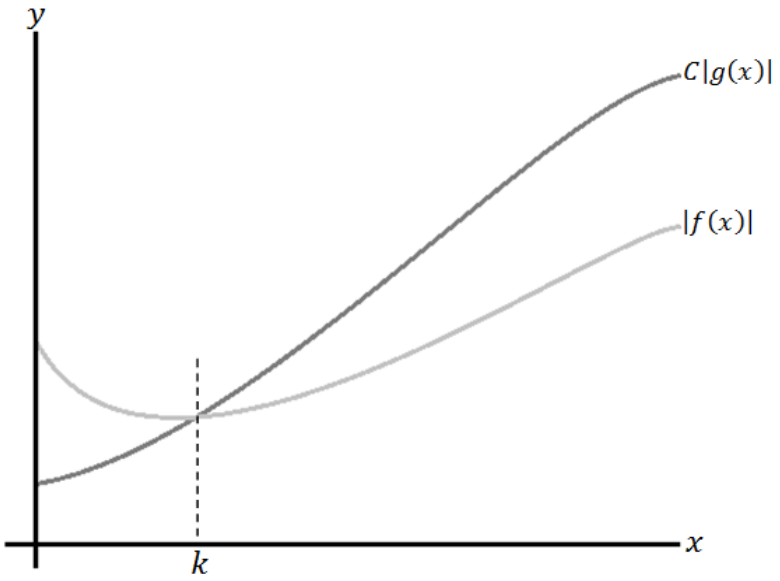

Figura 8: función $f\ x\ \in \mathcal{O}\ g\ x$

Cuando $f\ x\ \in \mathcal{O}\ g\ x$ y $h\ x$ es una función mayor en valor absoluto que $g\ x$, se cumple que $f\ x\ \in \mathcal{O}\ h\ x$. Para la relación $f\ x\ \in \mathcal{O}\ g\ x$, la función $g\ x$ debe elegirse lo más pequeña posible.

En una función polinomial, el término de mayor grado de un polinomio domina su crecimiento, por lo tanto un polinomio de grado n o inferior es $\mathcal{O}\ x^n$; sea $f\ x\ = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0; a_0, a_1, \ldots, a_{n-1}, a_n \in \mathbb{R}$:

$$f\ x\ = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$$

$$f\ x\ \leq a_n\ x^n + a_{n-1}\ x^{n-1} + \cdots + a_1\ x + a_0$$

$$f\ x\ = x^n\ a_n + a_{n-1}\ /x + \cdots + a_1\ /x^{n-1} + a_0\ /x^n$$

$$f\ x\ \leq x^n\ a_n + a_{n-1} + \cdots + a_1 + a_0$$

$$\therefore f\ x\ \leq Cx^n; C = a_n + a_{n-1} + \cdots + a_1 + a_0\ , k > 1$$

se demuestra que $f\ x\ \in \mathcal{O}\ x^n$.

Por lo general, un algoritmo consiste en dos o más subprocesos separados, para obtener una estimación del número de operaciones requeridas por el algoritmo, se necesita calcular el número de operaciones en cada subproceso y combinar éste cálculo. Frecuentemente estos cálculos consisten en la suma y el producto de dos funciones.

Sean dos funciones $f_1\ x\ \in \mathcal{O}\ g_1\ x$ y $f_2\ x\ \in \mathcal{O}\ g_2\ x$, entonces hay cuatro constantes C_1, C_2, k_1 y k_2 tales que:

$$f_1\ x\ \ \leq C_1\ g_1\ x\ \ \text{cuando}\ x > k_1\ \text{y}$$

$$f_2\ x\ \ \leq C_2\ g_2\ x\ \ \text{cuando}\ x > k_2$$

por lo tanto:

$$f_1 + f_2\ \ x\ \ = f_1\ x\ + f_2\ x$$

$$f_1 + f_2\ \ x\ \ \leq f_1\ x\ \ + f_2\ x$$

cuando x es mayor que k_1 y k_2 se deduce que :

$$f_1\ x\ \ + f_2\ x\ \ \leq C_1\ g_1\ x\ \ + C_2\ g_2\ x$$

$$f_1\ x\ \ + f_2\ x\ \ \leq C_1\ g\ x\ \ + C_2\ g\ x$$

$$f_1\ x\ \ + f_2\ x\ \ \leq\ C_1{+}C_2\ \ g\ x$$

$$f_1\ x\ \ + f_2\ x\ \ \leq C\ g\ x$$

donde $C = C_1{+}C_2$, $g\ x\ = \max\ g_1\ x\ ,\ g_2\ x$ cuando $x > k$, $k = \max\ k_1, k_2$; $\max\ a, b$ denota el mayor de a o b. Finalmente se concluye:

Sean dos funciones $f_1\ x\ \in \mathcal{O}\ g_1\ x$ y $f_2\ x\ \in \mathcal{O}\ g_2\ x$, entonces $f_1 + f_2\ \ x\ \in \mathcal{O}\ \max\ g_1\ x\ ,\ g_2\ x$.

En el caso de que $f_1\ x\ \in \mathcal{O}\ g\ x$ y $f_2\ x\ \in \mathcal{O}\ g\ x$, entonces $f_1 + f_2\ \ x\ \in \mathcal{O}\ g\ x$ puesto que el $\max\ g\ x\ ,\ g\ x\ = g\ x$.

De manera análoga, si existen dos funciones $f_1\ x\ \in \mathcal{O}\ g_1\ x$ y $f_2\ x\ \in \mathcal{O}\ g_2\ x$, entonces:

$$f_1 f_2\ \ x\ \ = f_1\ x\ \ f_2\ x$$

$$f_1 f_2\ \ x\ \ \leq C_1\ g_1\ x\ \ C_2\ g_2\ x$$

$$f_1 f_2\ \ x\ \ \leq C_1 C_2\ \ g_1 g_2\ \ x$$

$$f_1 f_2\ \ x\ \ \leq C\ \ g_1 g_2\ \ x$$

donde $C = C_1 C_2, k = \max\ k_1, k_2$, cuando $x > k$. Se concluye:

Sean dos funciones $f_1\ x\ \in \mathcal{O}\ g_1\ x$ y $f_2\ x\ \in \mathcal{O}\ g_2\ x$, entonces $f_1 f_2\ \ x\ \in \mathcal{O}\ g_1\ x\ g_2\ x$.

3.1.1 Complejidad computacional de las operaciones comunes en los algoritmos

La suma de los primeros n enteros positivos es $\mathcal{O}\ n^2$; debido a que cada término de la suma no es mayor que n:

$$1 + 2 + \cdots + n \leq n + n + \cdots + n$$

$$1 + 2 + \cdots + n \leq n^2$$

se demuestra que $1 + 2 + \cdots + n \in \mathcal{O}\ n^2$, cuando los testigos son $C = k = 1$.

La función factorial es $\mathcal{O}\ n^n$; la función factorial $f\ n\ = n!$ se define como $f\ n\ = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$, debido a que cada término del producto no es mayor que n:

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$$

$$n! \leq n \cdot n \cdot n \cdot \ldots \cdot n$$

$$n! \leq n^n$$

se demuestra que $n! \in \mathcal{O}\ n^n$, cuando los testigos son $C = k = 1$.

La función $f\ n\ = \log n!$ es $\mathcal{O}\ n \log n$; al aplicar logaritmos en ambos miembros de la desigualdad $n! \leq n^n$:

$$\log n! \leq \log n^n$$

$$\log n! \leq n \log n$$

se demuestra que $\log n! \in \mathcal{O}\ n \log n$, cuando los testigos son $C = k = 1$.

La desigualdad $n < 2^n$ es verdadera para cualquier entero positivo n, con esta desigualdad se puede concluir que $n \in \mathcal{O}\ 2^n$, cuando los testigos son $C = k = 1$. La demostración de la desigualdad $n < 2^n$ se realiza mediante inducción matemática:

1. Proposición $P\ n\ : n < 2^n$.
2. Comprobación para $P\ 1\ : 1 < 2^1 = 2$.
3. Hipótesis inductiva $P\ k\ : k < 2^k$.
4. Tesis inductiva $P\ k + 1\ : k + 1 < 2^{k+1}$.
5. Demostración de la tesis en base a la hipótesis: $k + 1 < 2^k + 1 \leq 2^k + 2^k = 2^{k+1}$.

Al aplicar logaritmos base 2 en cada miembro de la desigualdad $n < 2^n$ se cumple que:

$$\log_2 n < n$$

se demuestra que $\log_2 n \in \mathcal{O}\ n$, cuando los testigos son $C = k = 1$.

Cuando se tienen logaritmos en una base distinta a 2, $\log_b n$ también es $\mathcal{O}\ n$ debido a que:

$$\log_b n = \frac{\log n}{\log b} < \frac{n}{\log b}$$

se demuestra que $\log_b n \in \mathcal{O}\ n$, cuando los testigos son $C = \frac{1}{\log b}$, $k = 1$.

3.2 Definición matemática de un sistema criptográfico asimétrico

Una *función unidireccional* $f: M \rightarrow C$ se define como una función invertible, de modo que es fácil[9] calcular $f\ m\ = c$, mientras que es difícil[10] calcular $f^{-1}\ c\ = m$. Se dice que una función unidireccional es una *función unidireccional tramposa* si puede ser invertida fácilmente cuando se conoce alguna información adicional extra, dicha información se conoce como *trampa*.

Matemáticamente, un sistema criptográfico asimétrico se define como una familia de funciones unidireccionales tramposas f_k, para cada clave k que pertenece al conjunto K de todas las claves utilizables, tal que la trampa $t(k)$ sea fácil de obtener. Para cada $k \in K$ se debe desarrollar un algoritmo eficiente que permita calcular f_k, pero que sea intratable la determinación de k y $t\ k$.

Para su implementación, dada una familia de funciones unidireccionales tramposas, cada usuario U elige una clave aleatoria $u \in K$ y publica E_u que permite calcular f_u; E_u es su *clave pública*, mientras que la trampa $t(u)$, necesaria para invertir f_u, es su *clave privada*.

Si un usuario A desea enviar un mensaje m a otro usuario B, mira la clave pública de B (E_b), y transmite $f_b\ m\ = c$ a B. Como B es el único capaz de invertir f_B, es el único que puede recuperar el mensaje m: $f_b^{-1}\ c\ = f_b^{-1}\ f_b\ m\ = m$.

[9] En criptografía, el término "fácil" indica que el cómputo se puede realizar en un espacio de tiempo tolerable.

[10] En criptografía, el término "difícil" indica que no puede computarse en un tiempo aceptable utilizando el mejor algoritmo conocido ni la mejor tecnología disponible.

3.3 El problema del logaritmo discreto

En la actualidad no se ha demostrado aun la existencia de funciones unidireccionales tramposas; sin embargo, existen dos funciones cercanas a serlo. La primera de ellas es el producto de números enteros, cuya inversa es la factorización del número obtenido.

Dado un entero compuesto n, el problema de la factorización es encontrar enteros positivos p, q, tales que $pq = n$. El problema puede ser resuelto en un tiempo $\mathcal{O}\left(\overline{n} \cdot \log n\right)^2$ usando *división por tanteo*, esto es, revisando exhaustivamente si p divide a n para $p = 2, \ldots, \overline{n}$. Este método requiere $\overline{n}$ divisiones, cada una tomando un tiempo $\log n^2$. Esto siempre sucede porque aunque el factor primo más grande de n puede ser tan grande como $n/2$, el factor primo más pequeño de n puede ser a lo mucho $\overline{n}$.

La segunda es la exponenciación discreta, cuya inversa es el logaritmo discreto. Sea $(G, *)$ un *grupo cíclico finito*[11] de orden n (con n elementos) y donde $*$ es el operador multiplicación, entonces existe un generador $g \in G$ tal que:

$$G = \{g^0, g^1, g^2, \ldots, g^{n-1}\}$$

Puesto que $(G, *)$ es cíclico, para toda $h \in G$ existe una única $k \in \mathbb{Z}_n^*$ tal que $g^k = h$, llamamos a esta k el *logaritmo de h con respecto a g* y se escribe $k = log_g\, h$. Resolver este problema consiste en encontrar un método computacionalmente eficiente que encuentre logaritmos discretos en el grupo dado.

La mayoría de los algoritmos para encontrar logaritmos discretos en un grupo dado calculan g^0, g^1, g^2 hasta que k es encontrada. Este método toma $\mathcal{O}(n)$ multiplicaciones, donde n es el orden de g, lo que rápidamente llegar a ser impráctico a medida que se aumenta el orden de g.

Cuando se trata de determinar un algoritmo eficiente que permita el cálculo de k, se le llama *problema del logaritmo discreto generalizado*. Si el grupo G es el grupo multiplicativo de los enteros módulo un número primo ($\mathbb{Z}_p^*$), el problema se le conoce simplemente como el *problema del logaritmo discreto*. Actualmente no se conocen algoritmos para factorizar o calcular logaritmos discretos en tiempo polinomial.

[11] Un grupo cíclico, es un grupo abeliano que puede ser generado por un solo elemento; es decir, $\exists\, g \in G$ (llamado *generador* de G), tal que todo elemento que pertenece G, puede ser expresado como una potencia de g.

3.3.1 El problema del logaritmo discreto para una curva elíptica

El problema del logaritmo discreto para una curva elíptica, consiste en encontrar una k para el cual $P_2 = kP_1$, donde P_1 y P_2 son dos puntos sobre una curva elíptica $E\ \mathbb{Z}_p$ sobre un campo finito. Se asume que k existe, debido a que el grupo de curvas elípticas es cíclico y P_2 lo genera.

Al realizar una búsqueda exhaustiva de k dados P_1 y P_2, el método tiene una complejidad $\mathcal{O}\ n$, donde n es el orden de P_1, lo que lo hace un problema intratable cuando el orden de P_1 es grande.

3.4 Diferencias y complejidad temporal de las operaciones entre los grupos $E\ \mathbb{Z}_p$ y $\mathbb{Z}_p^*$

La operación de grupo para una curva elíptica $E\ \mathbb{Z}_p$ es la suma, en contraste a la operación de grupo en $\mathbb{Z}_p^*$ que es la multiplicación. La siguiente tabla resume las diferencias de los elementos, operaciones, notaciones y problema del logaritmo discreto entre ambos grupos:

Concepto\Grupo	$\mathbb{Z}_p^*$	$E\ \mathbb{Z}_p$
Elementos de grupo.	Números enteros positivos: $1, 2, \dots, p-1$.	Coordenadas (x, y) de puntos sobre una curva elíptica $E\ \mathbb{Z}_p\ \cup\ \infty$.
Operación de grupo.	Multiplicación módulo p.	Suma de puntos sobre $E\ \mathbb{Z}_p$.
Notación.	Elementos: g, h. Multiplicación: gh. Inverso: g^{-1}. División: g/h. Exponenciación: g^k.	Elementos: P_1, P_2. Suma: $P_1 + P_2$. Negativo: $-P_1$. Sustracción: $P_1 - P_2$. Múltiplo: kP_1.
Problema del logaritmo discreto.	Dado $g\ \in\ \mathbb{Z}_p^*$ y $h = g^k \bmod p$, encontrar k.	Dado $P_1 \in E\ \mathbb{Z}_p$ y $P_2 = kP_1$, encontrar k.

Tabla 2: Diferencias de los elementos, operaciones, notaciones y problema del logaritmo discreto entre el grupo aditivo y el grupo multiplicativo de los números enteros módulo un número primo p.

Sean a y b dos enteros no negativos, cada uno menor o igual a n, las operaciones matemáticas que utilizan los algoritmos de cifrado asimétrico que emplean estos grupos y su complejidad son las siguientes:

Operación		Complejidad
Adición	$a + b$	$\mathcal{O}\ \log a + \log b\ = \mathcal{O}\ \log n$
Sustracción	$a - b$	$\mathcal{O}\ \log a + \log b\ = \mathcal{O}\ \log n$
Multiplicación	$a \cdot b$	$\mathcal{O}\ \ \log a\ \ \log b\ \ = \mathcal{O}\ \ \log n\ ^2$
División	$a = qb + r$	$\mathcal{O}\ \ \log q\ \ \log b\ \ = \mathcal{O}\ \ \log n\ ^2$
Adición modular	$a + b\ \mod n$	$\mathcal{O}\ \log n$
Sustracción modular	$a - b\ \mod n$	$\mathcal{O}\ \log n$
Multiplicación modular	$a \cdot b\ \mod n$	$\mathcal{O}\ \ \log n\ ^2$
Inversión modular	$a^{-1}\ \mod n$	$\mathcal{O}\ \ \log n\ ^2$
Exponenciación modular	$a^m\ \mod n, m < n$	$\mathcal{O}\ \ \log n\ ^3$

Tabla 3: Complejidades computacionales de algunas operaciones comunes en los algoritmos de los sistemas criptográficos asimétricos.

3.5 Algoritmo Diffie – Hellman para el intercambio de claves

El algoritmo Diffie – Hellman, también llamado *acuerdo de claves exponencial*, fue desarrollado por Whitfield Diffie y Martin E. Hellman en 1976 y publicado en el artículo *New directions in cryptography*[16]. El algoritmo permite a dos usuarios intercambiar una clave secreta a través de un canal no seguro.

El algoritmo es el siguiente:

1. Dos usuarios A y B seleccionan públicamente un grupo multiplicativo finito G, y un elemento $\alpha \in G$.
2. El usuario A genera un número aleatorio a tal que $0 < a < p - 1$, calcula α^a en G y transmite este elemento al usuario B.
3. De la misma manera el usuario B genera un número aleatorio b tal que $0 < b < p - 1$, calcula α^b en G y transmite este elemento al usuario A.
4. El usuario A recibe α^b y calcula $\alpha^{b\ a} \in G$.
5. Igualmente, el usuario B recibe α^a y calcula $\alpha^{a\ b} \in G$.

Ahora los usuarios A y B poseen un elemento común y secreto del grupo G: $\alpha^{ab} = \alpha^{ba}$. Un escucha (espía) S, podría conocer G, p, α^a, α^b y resolver el problema del logaritmo discreto para calcular el elemento α^{ab}, lo que hasta ahora es un problema intratable utilizando el mejor algoritmo conocido y la mejor tecnología disponible.

Para la implementación[12] de este algoritmo se emplearon los siguientes parámetros:

- Grupo multiplicativo finito G: $\mathbb{Z}_{11}^*$, orden del grupo $n = 5$.
- $\alpha \in G$: 5.
- Número aleatorio a: es generado aleatoriamente por el programa en cada ejecución.
- Número aleatorio b: es generado aleatoriamente por el programa en cada ejecución.

La salida del programa fue la siguiente:

```
Número público transmitido por el usuario A al usuario B: 3

Número público transmitido por el usuario B al usuario A: 5

Número secreto calculado por el usuario A: 3

Número secreto calculado por el usuario B: 3
```

3.5.1 Algoritmo Diffie – Hellman para el intercambio de claves empleando curvas elípticas

A continuación se presenta el algoritmo Diffie – Hellman en el cual el grupo multiplicativo finito G es remplazado por una curva elíptica $E\ \mathbb{Z}_p$.

Sean dos usuarios A y B, deben acordar públicamente un campo finito $\mathbb{Z}_p$ y una curva elíptica $E\ \mathbb{Z}_p$ sobre él. Los usuarios A y B eligen públicamente un punto P_1 sobre $E\ \mathbb{Z}_p$, una vez elegido el punto P_1, se genera la clave de la siguiente forma:

1. Sea n el orden del grupo, el usuario A elige secretamente un número aleatorio a tal que $0 < a < n$.
2. El usuario A calcula $a \cdot P_1$ sobre $E\ \mathbb{Z}_p$ y lo envía al usuario B.
3. Análogamente el usuario B selecciona un número aleatorio b.
4. El usuario B calcula $b \cdot P_1$ sobre $E\ \mathbb{Z}_p$ y lo envía al usuario A.
5. El usuario A calcula el punto $P_2 = a \cdot b \cdot P_1$ que pertenece a la curva elíptica $E\ \mathbb{Z}_p$.
6. El usuario B calcula el mismo punto $P_2 = b \cdot a \cdot P_1$.

[12] El código fuente del programa en lenguaje Java se encuentra en el apéndice D.

Ahora los usuarios A y B poseen un punto secreto y común sobre $E\ \mathbb{Z}_p$, el punto $P_2 = a \cdot b \cdot P_1 = b \cdot a \cdot P_1$. Un escucha podría conocer los puntos P_1, $a \cdot P_1$ y $b \cdot P_1$, del protocolo anterior, pero no podría calcular el punto P_2 en un tiempo razonable, debido al problema del logaritmo discreto para una curva elíptica.

Para la implementación[13] de este algoritmo se emplearon los siguientes parámetros:

- Curva elíptica $E\ \mathbb{Z}_{23} : y^2 = x^3 + Ax + B$; $A = 1$, $B = 1$, orden del grupo $n = 28$.
- Punto público $P_1 = \ 3, 10\ $.
- Número secreto $a = 15$.
- Número secreto $b = 19$.

La salida del programa fue la siguiente:

```
Punto público transmitido por el usuario A al usuario B: (1, 16)

Punto público transmitido por el usuario B al usuario A: (0, 22)

Punto secreto calculado por el usuario A: (9, 16)

Punto secreto calculado por el usuario B: (9, 16)
```

3.6 Sistema criptográfico asimétrico ElGamal

Este sistema criptográfico asimétrico fue descrito por Taher ElGamal en 1985[17], el cifrado se define a continuación. Se establece un número primo grande p el cual es público. También es pública una *raíz primitiva*[14] $g \bmod p$ tal que $1 < g < p$. Un usuario A que desee participar en la comunicación escoge un a_A secreto tal que $0 < a_A < p - 1$ y publica $b_A = g^{a_A} \bmod p$. Cuando un usuario B desea enviar un mensaje secreto M tal que $0 < M < p$ al usuario A, elige una k aleatoria tal que $0 < k < p - 1$ y envía al usuario A el par:

$$C = \ g^k \bmod p, M{b_A}^k \bmod p$$

El texto plano M es cifrado al multiplicarlo por ${b_A}^k$ en la segunda componente de C. Obsérvese que ${b_A}^k \equiv \ \ g^{a_A\ k} \equiv g^{a_A k} \bmod p$. La primer componente de C proporciona

[13] El código fuente del programa en lenguaje Java se encuentra en el apéndice G.

[14] Si g es una raíz primitiva módulo m, entonces el mcd $g, m\ = 1$.

un indicio para descifrar M de la segunda componente de C, sin embargo, solo le es útil al usuario A. Solamente el usuario A puede calcular $g^{k\ a_A} \equiv g^{a_A k} \bmod p$. Si el inverso modular de este número es multiplicado por la segunda componente de C, es posible recuperar M:

$$g^{a_A k\ -1}\ {Mb_A}^k \ \equiv\ g^{a_A k\ -1}\ Mg^{a_A k} \ \equiv M \bmod p$$

Un espía quien pudiera resolver el problema del logaritmo discreto módulo p, podría calcular M de C y publicar los datos sin conocer a_A. La primer componente de C es $h = g^k \bmod p$. Este número y ${Mb_A}^k \bmod p$ son escuchados por el espía. El espía conoce g y p porque estos números son públicos. El también podría obtener la clave pública b_A como el usuario B lo hace. Él podría resolver el problema del logaritmo discreto $g^k \equiv h \bmod p$ para k y calcular:

$$M{b_A}^k \quad {b_A}^{k\ -1} \equiv M \bmod p$$

Para la implementación[15] de este sistema se emplearon los siguientes parámetros:

- Campo finito: $\mathbb{Z}_{127}^*$, orden del grupo $n = 31$.
- Raíz primitiva $g\ =\ 2$.
- $a_A = 3$.
- Alfabeto P: equivalente numérico decimal de los 95 caracteres imprimibles del ASCII (caracteres 32 al 126).

La salida del programa fue la siguiente:

```
Ingrese cadena: CONFIDENCIALIDAD

Cadena cifrada: 16 112 4 103 2 116 8 26 4 100 2 36 2 44 2 116 4 97 1 73 8
6 4 38 8 38 4 34 32 3 64 72

Ingrese cadena a descifrar: 16 112 4 103 2 116 8 26 4 100 2 36 2 44 2 116
4 97 1 73 8 6 4 38 8 38 4 34 32 3 64 72

Cadena descifrada: CONFIDENCIALIDAD
```

[15] El código fuente del programa en lenguaje Java se encuentra en el apéndice E.

3.6.1 Sistema criptográfico asimétrico ElGamal empleando curvas elípticas

El sistema criptográfico asimétrico análogo a ElGamal empleando curvas elípticas se define como sigue. Considérese una curva elíptica $E\ \mathbb{Z}_p$ y un punto P_0 sobre $E\ \mathbb{Z}_p$, todos estos datos son públicos. Cada usuario A que desee participar en la comunicación escoge un a_A tal que $0 < a_A < p$ y publica $P_A = a_A P_0$ sobre $E\ \mathbb{Z}_p$. Cuando un usuario B desea enviar un mensaje secreto M al usuario A, escoge una k aleatoria tal que $0 < k < p$ y envía al usuario el par:

$$C = kP_0, kP_A + P$$

El texto plano P es cifrado al sumar el punto kP_A en la segunda componente de C, $kP_A = k\ a_A P_0 = ka_A\ P_0$. La primer componente de C proporciona un indicio para descifrar P de la segunda componente de C, sin embargo, solo le es útil al usuario A. Solamente el usuario A conoce la clave secreta a_A y puede calcular $a_A\ kP_0 = ka_A\ P_0$. Si este punto es sustraído de la segunda componente, P se puede recuperar así: $kP_A + P - ka_A\ P_0 = ka_A\ P_0 + P - ka_A\ P_0 = P$.

Un espía que pudiera resolver el problema del logaritmo discreto sobre $E\ \mathbb{Z}_p$ podría calcular P de C y publicar los datos sin conocer a_A como sigue. La primera componente de C es $P_1 = kP_0$. Este punto y $T = kP_A + P$ son observados por el espía. El espía conoce p, E y P_0 porque esta información es pública. El también puede obtener la clave pública P_A tal como lo hace el usuario B. El podría resolver el problema del logaritmo discreto para una curva elíptica $kP_0 = P_1$ para k y entonces calcular $T - kP_A = kP_A + P - kP_A = P$.

Para la implementación[16] de este sistema se emplearon los siguientes parámetros:

- Curva elíptica $E\ \mathbb{Z}_{97}: y^2 = x^3 + Ax + B;\ A = 1,\ B = 2,$ orden del grupo $n = 104$.
- $P_0 \in E\ \mathbb{Z}_p = 41{,}71$.
- $a_A = 2$.
- $k = 4$.
- Alfabeto P:

[16] El código fuente del programa en lenguaje Java se encuentra en el apéndice H.

(espacio) = (0, 14)
! = (0, 83)
" = (1, 2)
= (1, 95)
$ = (2, 20)
% = (2, 77)
& = (3, 41)
' = (3, 56)
(= (4, 19)
) = (4, 78)
* = (5, 36)
+ = (5, 61)
, = (7, 35)
- = (7, 62)
. = (9, 35)
/ = (9, 62)
0 = (12, 44)
1 = (12, 53)
2 = (14, 23)
3 = (14, 74)
4 = (15, 26)
5 = (15, 71)
6 = (18, 41)
7 = (18, 56)
8 = (22, 14)
9 = (22, 83)
: = (25, 36)
; = (25, 61)
< = (26, 12)
= = (26, 85)
> = (29, 48)
? = (29, 49)
@ = (30, 39)
A = (30, 58)
B = (32, 4)
C = (32, 93)
D = (39, 44)
E = (39, 53)
F = (40, 33)
G = (40, 64)
H = (41, 26)
I = (41, 71)
J = (42, 11)
K = (42, 86)
L = (43, 20)
M = (43, 77)
N = (44, 8)
O = (44, 89)
P = (45, 34)
Q = (45, 63)
R = (46, 44)
S = (46, 53)
T = (48, 16)
U = (48, 81)
V = (51, 28)
W = (51, 69)
X = (52, 20)
Y = (52, 77)
Z = (54, 34)
[= (54, 63)
\ = (57, 46)
] = (57, 51)
^ = (58, 28)
_ = (58, 69)
` = (59, 24)
a = (59, 73)
b = (60, 25)
c = (60, 72)
d = (61, 8)
e = (61, 89)
f = (65, 45)
g = (65, 52)
h = (67, 36)
i = (67, 61)
j = (71, 32)
k = (71, 65)
l = (72, 39)
m = (72, 58)
n = (73, 5)
o = (73, 92)
p = (75, 14)
q = (75, 83)
r = (76, 41)
s = (76, 56)
t = (77, 18)
u = (77, 79)
v = (78, 37)
w = (78, 60)
x = (81, 35)
y = (81, 62)
z = (85, 28)
{ = (85, 69)
| = (86, 42)
} = (86, 55)
~ = (89, 8)

La salida del programa fue la siguiente:

```
Ingrese cadena: CONFIDENCIALIDAD

Cadena cifrada: 42 11 5 61 42 11 22 14 42 11 67 61 42 11 18 41 42 11 57
51 42 11 14 23 42 11 81 62 42 11 67 61 42 11 5 61 42 11 57 51 42 11 72 39
42 11 15 71 42 11 57 51 42 11 14 23 42 11 72 39 42 11 14 23

Ingrese cadena a descifrar: 42 11 5 61 42 11 22 14 42 11 67 61 42 11 18
41 42 11 57 51 42 11 14 23 42 11 81 62 42 11 67 61 42 11 5 61 42 11 57 51
42 11 72 39 42 11 15 71 42 11 57 51 42 11 14 23 42 11 72 39 42 11 14 23

Cadena descifrada: CONFIDENCIALIDAD
```

3.7 Sistema criptográfico asimétrico Massey – Omura

En criptografía, el sistema criptográfico asimétrico de tres pasos para el envío de mensajes, es un sistema que permite a un usuario enviar un mensaje a un segundo usuario sin la necesidad de intercambiar o distribuir claves de cifrado. Es llamado protocolo de tres pasos porque el transmisor y el receptor intercambian tres mensajes cifrados.

El primer protocolo de tres pasos fue desarrollado por Adi Shamir[17] en 1980. El concepto básico del protocolo de tres pasos consiste en que cada usuario tiene una clave privada de cifrado y una clave privada de descifrado, ambos usuarios utilizan sus claves independientemente, primero para cifrar el mensaje, y después para descifrar el mensaje. El sistema criptográfico asimétrico Massey – Omura fue propuesto por James Massey y Jim K. Omura en 1982[18] como una probable mejora al protocolo de Shamir.

Supóngase que dos usuarios A y B tienen funciones de cifrado E_A y E_B y funciones de descifrado D_A y D_B, donde $E_A\ M\ = M^{e_A} \bmod p$ y $D_A\ C\ = C^{d_A} \bmod p$, además e_A y d_A son inversos modulares[18] $e_A d_A \equiv 1\ mod\ p-1$, etcétera. Ya que todas las funciones de cifrado y descifrado consisten de un módulo de exponenciación modulo fijo, todas conmutan, esto es, pueden ser de cualquier orden y dar el mismo resultado. Por ejemplo, $E_A\ D_B\ x\ = D_B\ E_A\ x$ para cada x debido a que ambos son $x^{e_A d_B} \equiv x^{d_B e_A} \bmod p$.

La clave pública es el primo común módulo p. Las claves privadas son todos los exponentes. Si el usuario A desea enviar un mensaje $0 < M < p$ al usuario B:

1. Envía $E_A\ M$ al usuario B.
2. El usuario B responde enviando $E_B\ E_A\ M$ al usuario A.
3. Después el usuario A envía $D_A\ E_B\ E_A\ M\ = E_B\ D_A\ E_A\ M\ = E_B\ M$ al usuario B.
4. El usuario B descifra el mensaje al aplicar D_B a $E_B\ M$.

Un espía podría ver los mensajes $r = E_A\ M$, $s = E_B\ E_A\ M$ y $t = D_A\ E_B\ E_A\ M\ = E_B\ D_A\ E_A\ M\ = E_B\ M$ pasados entre los usuarios A y B. Si el espía pudiera

[17] Adi Shamir (nacido en 1952) es un matemático israelí que ha hecho numerosas contribuciones a los campos de la criptografía y las ciencias de la computación.
[18] Ver apéndice A.

resolver el problema del algoritmo discreto, entonces el podría leer M de dos maneras. La primera, $s \equiv r^{e_B} \mod p$. El conoce s, r y p. Si él puede resolver para e_B, entonces puede calcular d_B utilizando el algoritmo extendido de Euclides[19]. Él podría calcular $M = t^{d_B} \mod p$.

La otra forma para leer M es usar la congruencia $s \equiv t^{e_A} \mod p$ para encontrar e_A resolviendo el problema del logaritmo discreto. Él podría calcular d_A de e_A por el algoritmo extendido de Euclides y encontrar $M = r^{d_A} \mod p$.

Para la implementación[20] de este algoritmo se emplearon los siguientes parámetros:

- $e_A = 9$, $d_A = 9$.
- $e_A = 11$, $d_A = 11$.
- Campo finito: $\mathbb{Z}_{41}^*$, orden del grupo $n = 13$.
- Alfabeto:

A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	2	3	4	5	6	7	8	9	10	11	12	13	14
O	P	Q	R	S	T	U	V	W	X	Y	Z	(espacio)	
15	16	16	18	19	20	21	22	23	25	25	26	27	

La salida del programa fue la siguiente:

```
Ingrese cadena: CONFIDENCIALIDAD

Cadena cifrada: 27 29 38 28 32 4 36 38 27 32 1 26 32 4 1 4

Ingrese cadena a descifrar: 27 29 38 28 32 4 36 38 27 32 1 26 32 4 1 4

Cadena descifrada: CONFIDENCIALIDAD
```

3.7.1 Sistema criptográfico asimétrico Massey – Omura empleando curvas elípticas

Considérese una curva elíptica $E\ \mathbb{Z}_p$ de orden n. Supóngase que dos usuarios A y B tienen funciones de cifrado E_A y E_B y funciones de descifrado D_A y D_B, donde $E_A\ P\ = e_A P$ sobre $E\ \mathbb{Z}_p$ y $D_A\ P\ = d_A P$, además $e_A d_A \equiv 1 \mod n$, etcétera. Ya que las

[19] Ver apéndice A.
[20] El código fuente del programa en lenguaje Java se encuentra en el apéndice F.

funciones de cifrado y descifrado son todas multiplicaciones escalares sobre una curva elíptica $E\ \mathbb{Z}_p$, todas ellas conmutan, es decir, pueden dar el mismo resultado. Por ejemplo, $E_A\ D_B\ P\ = D_B\ E_A\ P$ para cada P debido a que ambos son $e_A d_B P = d_B e_A P$.

Estás propiedades pueden ser utilizadas para un sistema criptográfico asimétrico, a continuación se presenta el sistema criptográfico asimétrico Massey – Omura empleando curvas elípticas.

Las claves públicas son $E\ \mathbb{Z}_p$, p y n. Las claves privadas son todos los multiplicadores e_A, d_a, etc. Si un usuario A desea enviar un mensaje $0 < M < p$ al usuario B, envía $E_A\ P$ al usuario B. El usuario B responde enviando $E_B\ E_A\ P$ al usuario A. Posteriormente el usuario A envía $D_A\ E_B\ E_A\ P\ = E_B\ D_A\ E_A\ P\ = E_B\ P$. El usuario B descifra el mensaje al aplicar $D_B\ E_B\ P$.

Un espía podría ver los mensajes $R = E_A\ P$, $S = E_B\ E_A\ P$ y $T = D_A\ E_B\ E_A\ P\ = E_B\ D_A\ E_A\ P\ = E_B\ P$ pasados entre los usuarios A y B. Si el espía pudiera resolver el problema del logaritmo discreto para los puntos sobre $E\ \mathbb{Z}_p$, entonces el podría leer el punto P de dos formas. La primera forma es $S = e_B R$, él conoce S, R, E, n y p; si el puede resolver para e_B, entonces él puede calcular d_B por el algoritmo extendido de Euclides[21] y obtener $P = d_B t$. La otra forma de encontrar P es usar la ecuación $S = e_A T$ para encontrar e_A; se puede calcular d_A a partir de e_A por el algoritmo extendido de Euclides y encontrar $P = d_A R$.

Para la implementación[22] de este sistema se emplearon los siguientes parámetros:

- $e_A = 5$.
- $d_A = 23$.
- $e_A = 7$.
- $d_A = 49$.
- Curva elíptica $E\ \mathbb{Z}_{127}$: $y^2 = x^3 + Ax + B$; $A = 3$, $B = 4$, orden del grupo $n = 114$.
- Alfabeto P:

[21] Ver apéndice A.
[22] El código fuente del programa en lenguaje Java se encuentra en el apéndice I.

(espacio)= (0, 2)
! = (0, 101)
" = (1, 27)
= (1, 76)
$ = (2, 11)
% = (2, 92)
& = (5, 12)
' = (5, 91)
(= (6, 49)
) = (6, 54)
* = (7, 33)
+ = (7, 70)
, = (8, 5)
- = (8, 98)
. = (10, 2)
/ = (10, 101)
0 = (11, 21)
1 = (11, 82)
2 = (12, 29)
3 = (12, 74)
4 = (14, 3)
5 = (14, 100)
6 = (15, 5)
7 = (15, 98)
8 = (16, 50)
9 = (16, 53)
: = (18, 15)
; = (18, 88)
< = (19, 15)
= = (19, 88)
> = (20, 37)
? = (20, 66)
@ = (21, 26)
A = (21, 77)
B = (24, 43)
C = (24, 60)
D = (31, 11)
E = (31, 92)
F = (33, 14)
G = (33, 89)
H = (34, 8)
I = (34, 95)
J = (35, 41)
K = (35, 62)
L = (37, 35)
M = (37, 68)
N = (38, 20)
O = (38, 83)
P = (39, 3)
Q = (39, 100)
R = (40, 26)
S = (40, 77)
T = (41, 48)
U = (41, 55)
V = (42, 26)
W = (42, 77)
X = (44, 6)
Y = (44, 97)
Z = (47, 12)
[= (47, 91)
\ = (48, 18)
] = (48, 85)
^ = (50, 3)
_ = (50, 100)
` = (51, 12)
a = (51, 91)
b = (58, 38)
c = (58, 65)
d = (66, 15)
e = (66, 88)
f = (67, 38)
g = (67, 65)
h = (70, 11)
i = (70, 92)
j = (71, 10)
k = (71, 93)
l = (72, 14)
m = (72, 89)
n = (76, 4)
o = (76, 99)
p = (77, 13)
q = (77, 90)
r = (78, 28)
s = (78, 75)
t = (79, 42)
u = (79, 61)
v = (80, 5)
w = (80, 98)
x = (81, 38)
y = (81, 65)
z = (83, 9)
{ = (83, 94)
| = (84, 35)
} = (84, 68)
~ = (85, 35)

La salida del programa fue la siguiente:

```
Ingrese cadena: CONFIDENCIALIDAD

Cadena cifrada: 8 14 64 121 2 116 1 70 2 76 2 36 8 22 64 105 64 56 8 38 4
96 8 50 1 73 8 18 32 3 8 18

Ingrese cadena a descifrar: 8 14 64 121 2 116 1 70 2 76 2 36 8 22 64 105
64 56 8 38 4 96 8 50 1 73 8 18 32 3 8 18

Cadena descifrada: CONFIDENCIALIDAD
```

Capítulo 4

Resultados

Capítulo 4

Resultados

Las siguientes tablas y gráficas muestran la comparación de la complejidad temporal y el crecimiento del tiempo de ejecución obtenido en cada algoritmo mediante la cota superior asintótica. Se analizó la complejidad en el cifrado, descifrado y la complejidad de rompimiento del problema del logaritmo discreto para cada algoritmo en su versión clásica y para curvas elípticas.

Este análisis determina el número de operaciones requeridas por el algoritmo a partir del tamaño de la entrada n. Además, se calculó el tiempo de ejecución de los algoritmos cuando el tamaño de la entrada n es lo suficientemente grande para romper (resolver) el problema del logaritmo discreto en un tiempo de 10 años, empleando una computadora capaz de realizar mil millones de operaciones sobre segundo.

El número de operaciones requeridas para que una computadora capaz de realizar 10^9 operaciones/segundo tarde 10 años en resolverlas, se obtuvo de la siguiente manera:

$$10 \text{ años } = 315,360,000 \text{ segundos}$$

$$\frac{x \ \ operaciones}{10^9 \ \ operaciones/segundo} = 315,360,000 \text{ segundos}$$

$$x = 315,360,000 \cdot 10^9 \text{ operaciones}$$

Como se demostró, resolver el problema del logaritmo discreto toma $\mathcal{O}\ n$ multiplicaciones (operaciones), donde n es el número total de bits necesarios para representar en notación binaria ordinaria el orden del grupo g empleado. Por lo tanto,

el orden n del grupo g empleado, deberá ocupar $315,360,000 \cdot 10^9$ bits en su representación binaria ordinaria:

$$\mathcal{O}\ n\ = 315,360,000 \cdot 10^9 \text{ operaciones}$$

$$\therefore n = 315,360,000 \cdot 10^9 \text{ bits}$$

4.1 Complejidad computacional y tiempo de rompimiento para el algoritmo Diffie – Hellman para el intercambio de claves

Versión del problema del logaritmo discreto empleado	Complejidad de rompimiento	Tiempo de rompimiento	Complejidad del algoritmo	Tiempo de ejecución del algoritmo [segundos]
Clásico	$\mathcal{O}\ n$	10 años	$\mathcal{O}\ \log n\ ^3$	$0.196 \cdot 10^{-3}$
Curvas elípticas	$\mathcal{O}\ n$	10 años	$\mathcal{O}\ \log n\ ^2$	$0.337 \cdot 10^{-5}$

Tabla 4: Complejidad computacional y tiempo de rompimiento para el algoritmo Diffie – Hellman para el intercambio de claves.

Para el algoritmo Diffie – Hellman, las siguientes gráficas muestran la comparación del crecimiento de las complejidades computacionales (figuras 9 y 10) y la comparación en el crecimiento del tiempo de ejecución (figura 11).

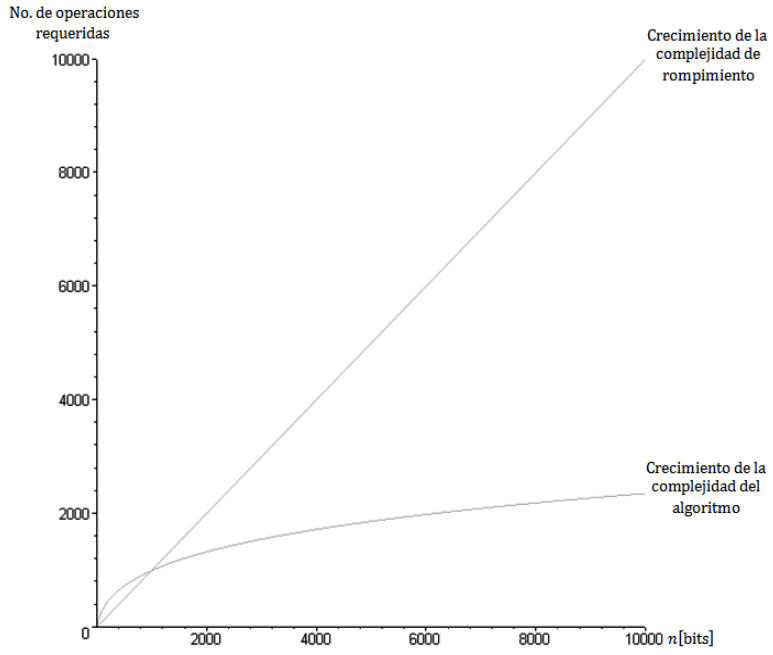

Figura 9: Crecimiento de las complejidades computacionales obtenidas para el algoritmo Diffie – Hellman para el intercambio de claves en la versión clásica.

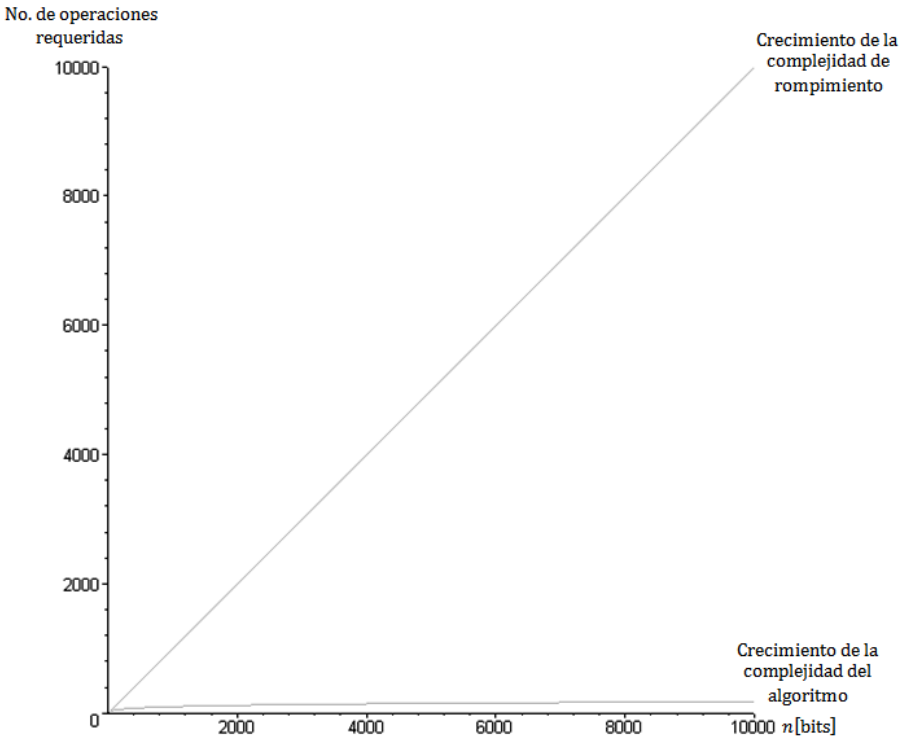

Figura 10: Crecimiento de las complejidades computacionales obtenidas para el algoritmo Diffie – Hellman para el intercambio de claves en la versión para curvas elípticas.

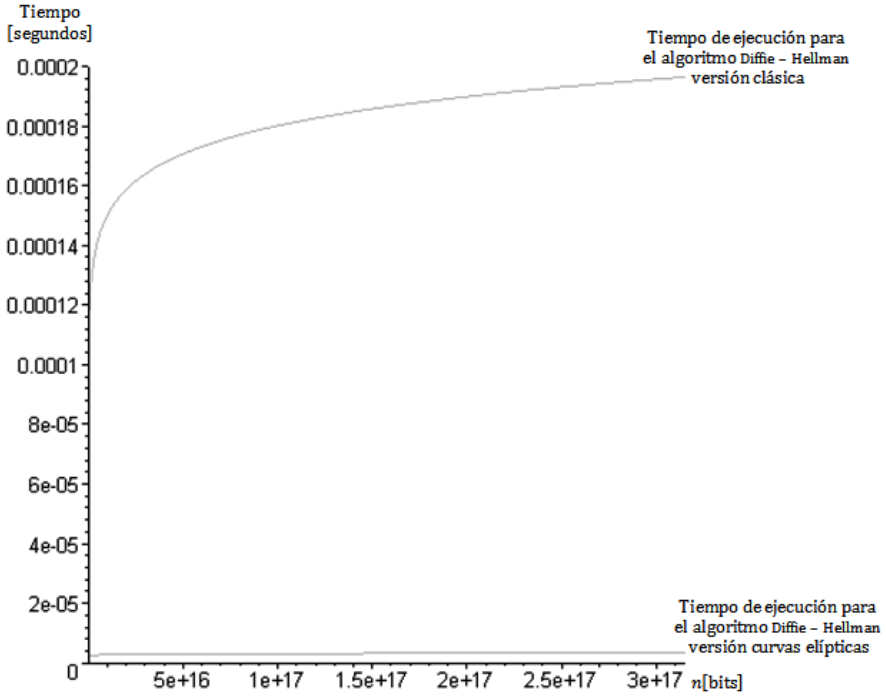

Figura 11: Crecimiento del tiempo de ejecución para el algoritmo Diffie – Hellman para el intercambio de claves.

4.2 Complejidad computacional y tiempo de rompimiento para el sistema criptográfico asimétrico ElGamal

Versión del problema del logaritmo discreto empleado	Complejidad de rompimiento	Tiempo de rompimiento	Complejidad de cifrado	Tiempo de cifrado [segundos]	Complejidad de descifrado	Tiempo de descifrado [segundos]
Clásico	$\mathcal{O}\ n$	10 años	$\mathcal{O}\ \log n\ ^3$	$0.196 \cdot 10^{-3}$	$\mathcal{O}\ \log n\ ^2$	$0.337 \cdot 10^{-5}$
Curvas elípticas	$\mathcal{O}\ n$	10 años	$\mathcal{O}\ \log n\ ^2$	$0.337 \cdot 10^{-5}$	$\mathcal{O}\ \log n\ ^2$	$0.337 \cdot 10^{-5}$

Tabla 5: Complejidad computacional y tiempo de rompimiento para el sistema criptográfico asimétrico ElGamal.

Para el sistema criptográfico ElGamal, las siguientes gráficas muestran la comparación del crecimiento de las complejidades computacionales (figuras 12 y 13), la comparación en el crecimiento del tiempo de cifrado (figura 14) y tiempo de descifrado (figura 15).

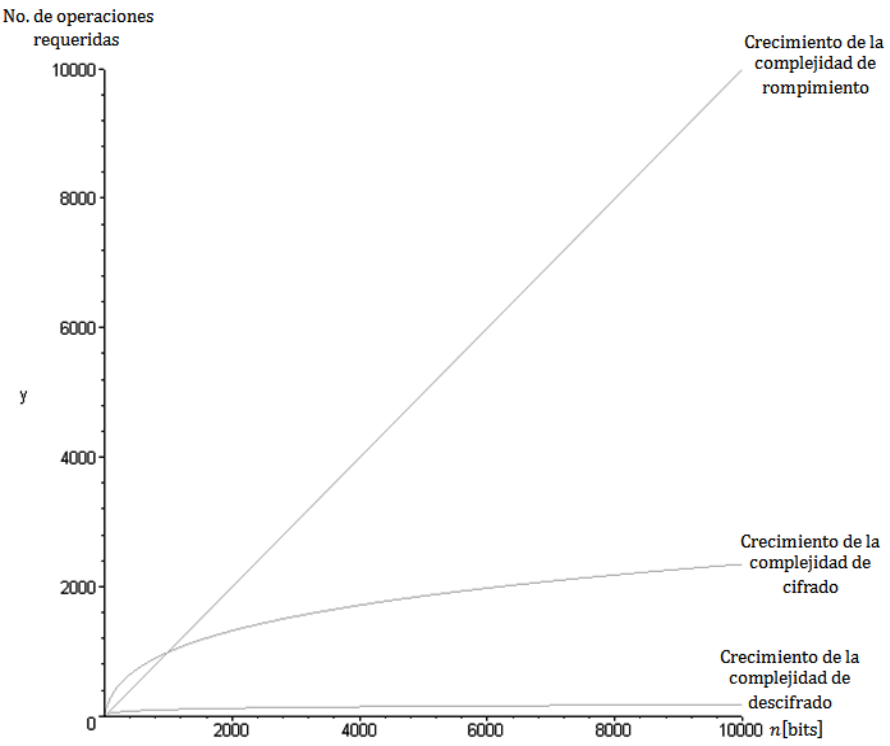

Figura 12: Crecimiento de las complejidades computacionales obtenidas para el sistema criptográfico asimétrico ElGamal en la versión clásica.

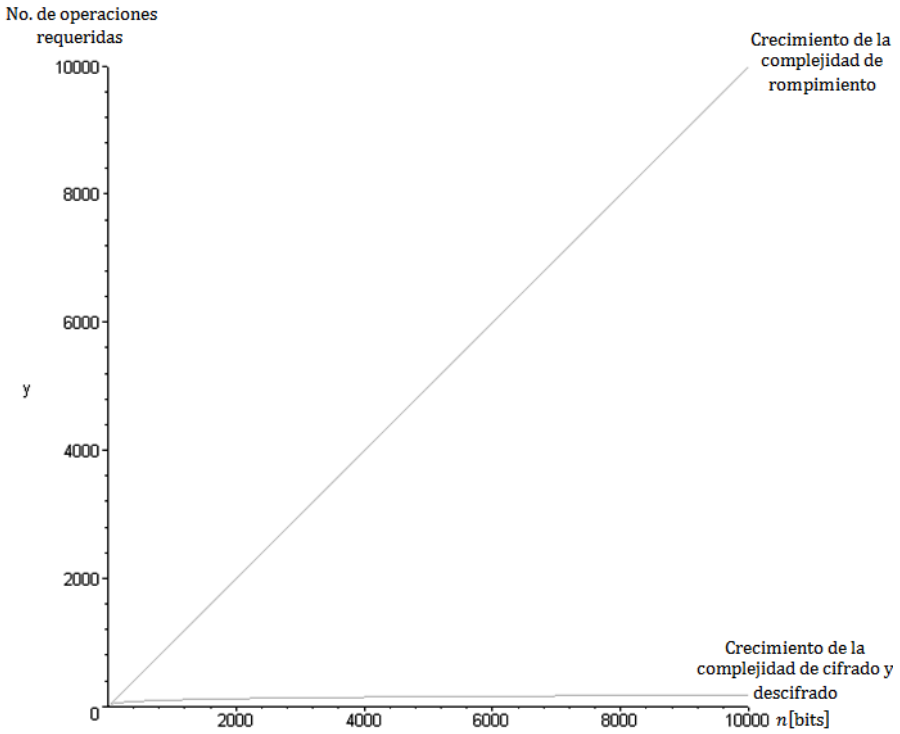

Figura 13: Crecimiento de las complejidades computacionales obtenidas para el sistema criptográfico asimétrico ElGamal en la versión para curvas elípticas.

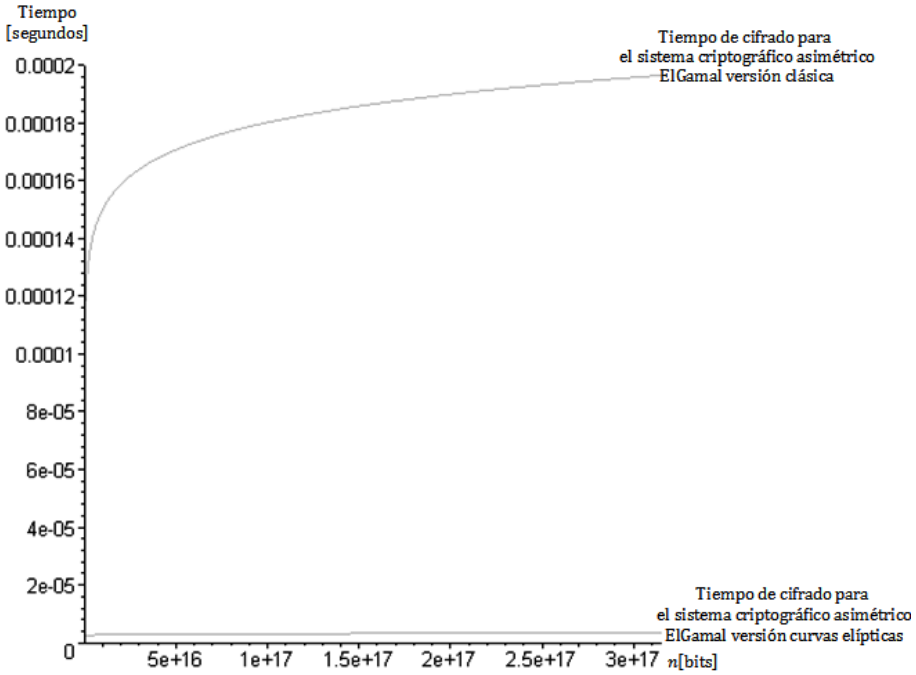

Figura 14: Crecimiento del tiempo de cifrado para el sistema criptográfico asimétrico ElGamal.

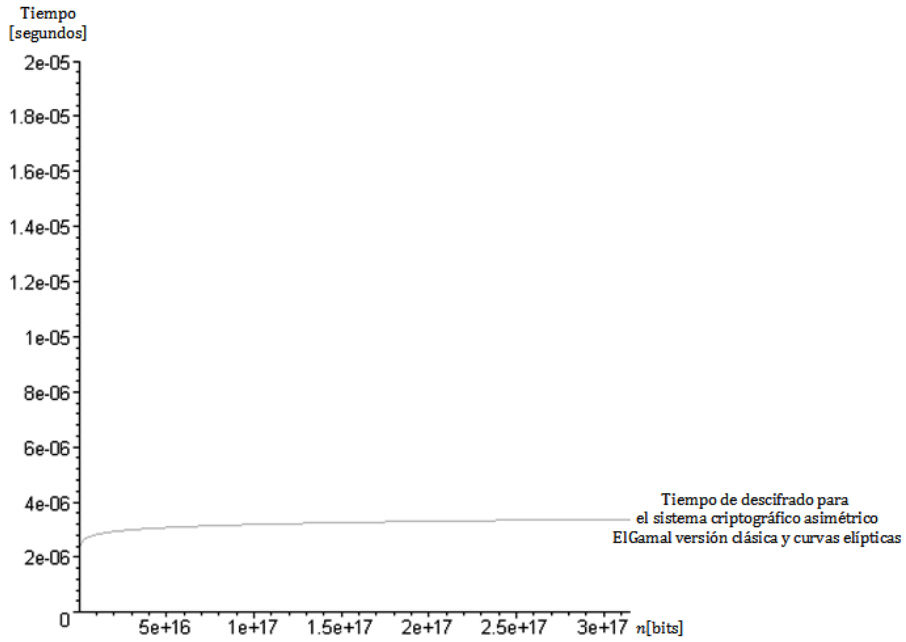

Figura 15: Crecimiento del tiempo de descifrado para el sistema criptográfico asimétrico ElGamal.

4.3 Complejidad computacional y tiempo de rompimiento para el sistema criptográfico asimétrico Massey – Omura

Versión del problema del logaritmo discreto empleado	Complejidad de rompimiento	Tiempo de rompimiento	Complejidad de cifrado	Tiempo de cifrado [segundos]	Complejidad de descifrado	Tiempo de descifrado [segundos]
Clásico	$\mathcal{O}\ n$	10 años	$\mathcal{O}\ \log n\ ^3$	$0.196 \cdot 10^{-3}$	$\mathcal{O}\ \log n\ ^3$	$0.196 \cdot 10^{-3}$
Curvas elípticas	$\mathcal{O}\ n$	10 años	$\mathcal{O}\ \log n\ ^2$	$0.337 \cdot 10^{-5}$	$\mathcal{O}\ \log n\ ^2$	$0.337 \cdot 10^{-5}$

Tabla 6: Complejidad computacional y tiempo de rompimiento para el sistema criptográfico asimétrico Massey – Omura.

Para el sistema criptográfico Massey – Omura, las siguientes gráficas muestran la comparación del crecimiento de las complejidades computacionales (figuras 16 y 17), la comparación en el crecimiento del tiempo de cifrado (figura 18) y tiempo de descifrado (figura 19).

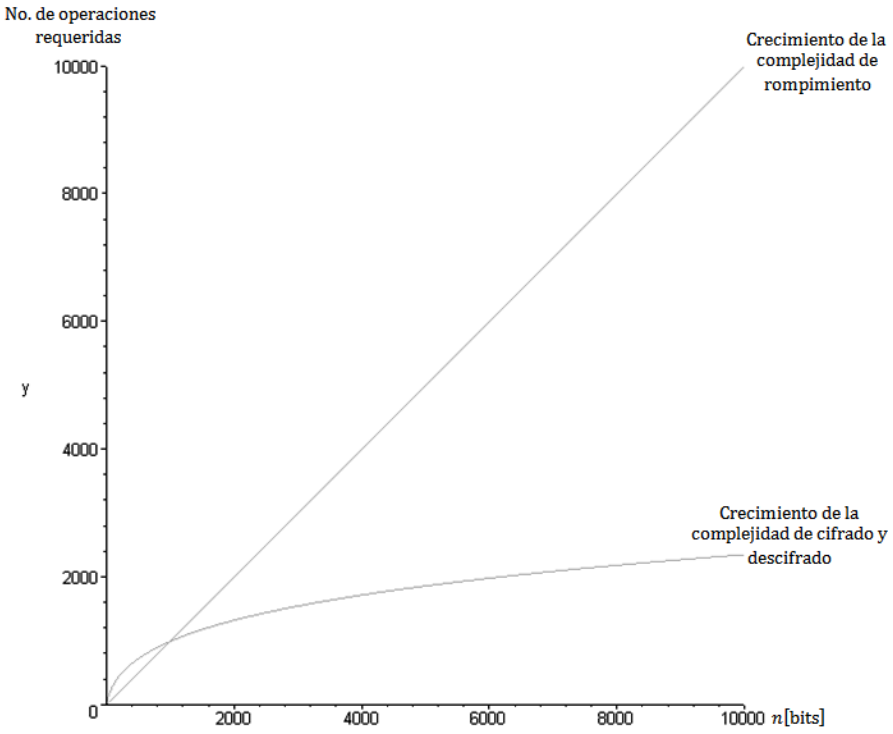

Figura 16: Crecimiento de las complejidades computacionales obtenidas para el sistema criptográfico asimétrico Massey – Omura en la versión clásica.

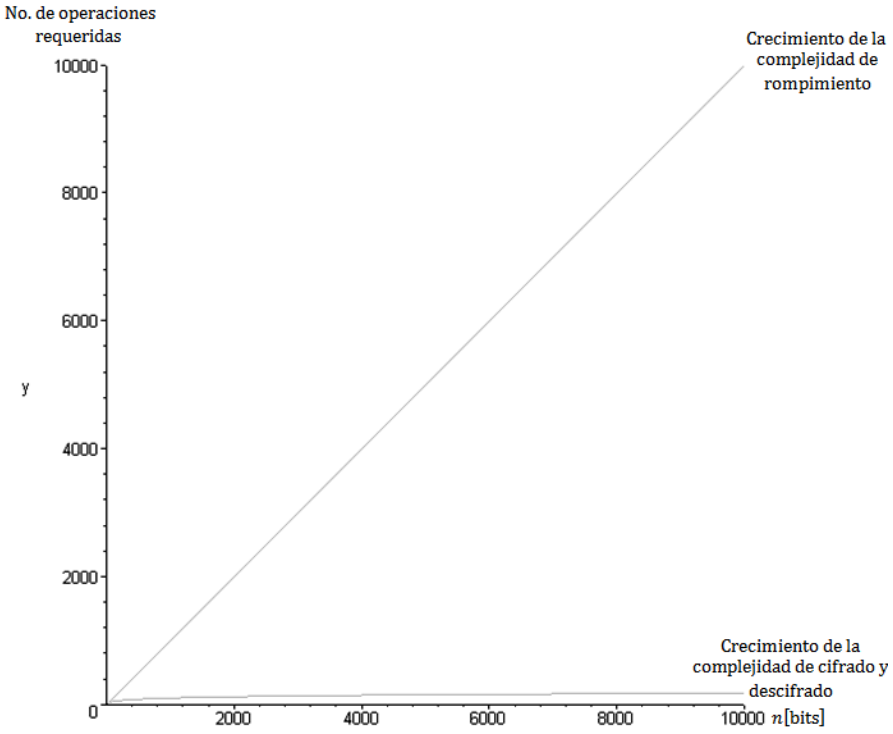

Figura 17: Crecimiento de las complejidades computacionales obtenidas para el sistema criptográfico asimétrico Massey – Omura en la versión para curvas elípticas.

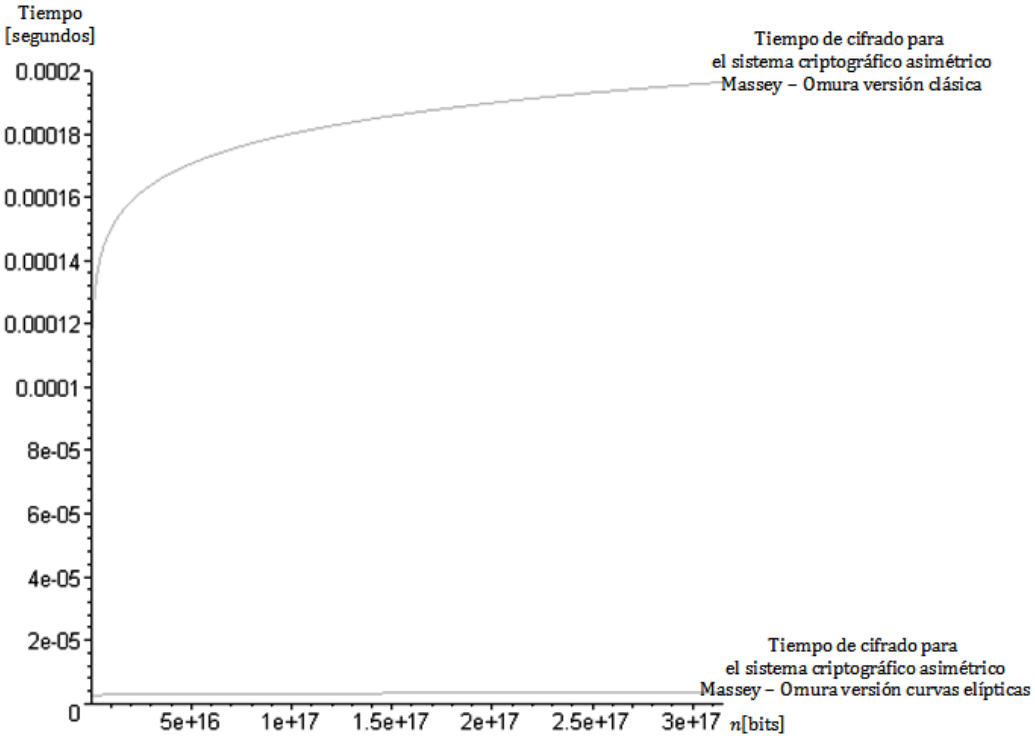

Figura 18: Crecimiento del tiempo de cifrado para el sistema criptográfico asimétrico Massey – Omura.

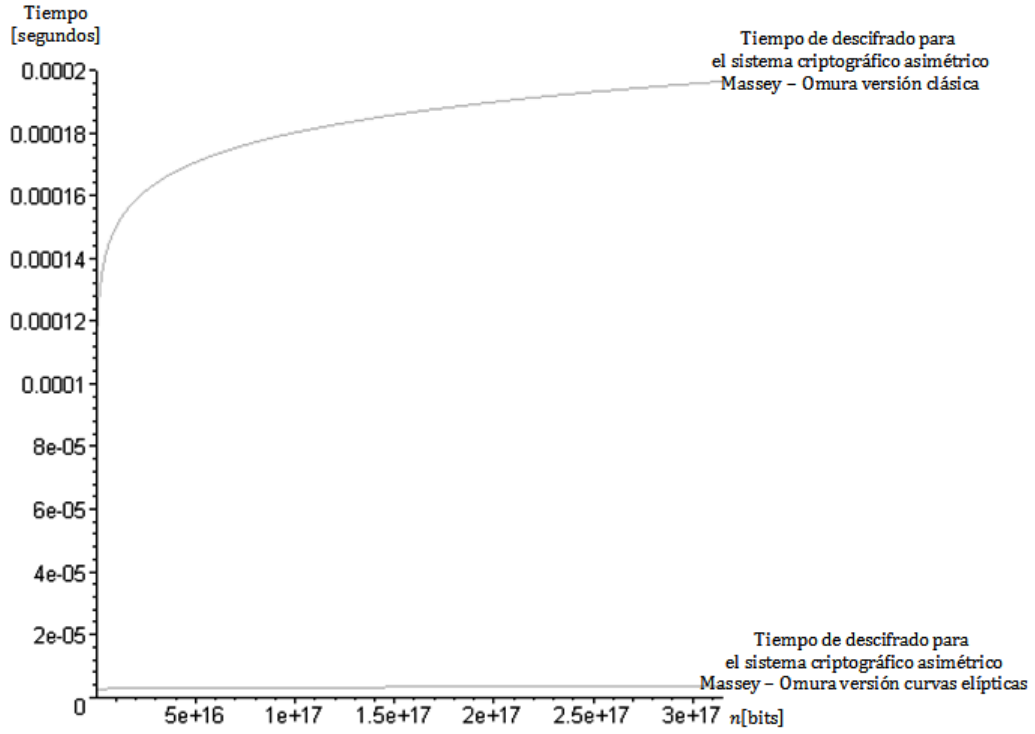

Figura 19: Crecimiento del tiempo de descifrado para el sistema criptográfico asimétrico Massey – Omura.

Conclusiones

Conclusiones

Se logró cumplir con los objetivos trazados, ya que fue posible corroborar y exponer las ventajas que presenta la criptografía de curva elíptica frente a los sistemas criptográficos asimétricos actuales.

El estudio de los fundamentos matemáticos de seguridad y el análisis del desempeño para los algoritmos de cifrado y descifrado revisados a lo largo de la presente obra, demostraron que efectivamente poseen el mismo nivel de seguridad en sus versiones para curvas elípticas, que para las versiones clásicas, además de que presentan mejor desempeño.

Los resultados despejaron las dudas en cuanto al desempeño de los algoritmos, pues aparentemente el número de operaciones empleadas para el cifrado y descifrado, es mayor que las necesarias para romper el sistema. Sin embargo, cuando el tamaño de la entrada crece, se evidencia que la complejidad para romper el sistema crece demasiado rápido, siendo solamente posible con el uso de equipos de cómputo actuales, el proceso de cifrado y descifrado, pero no el de rompimiento de su seguridad.

Los sistemas criptográficos de curva elíptica representan una mejor elección en entornos donde los recursos de procesamiento son limitados. Actualmente los dispositivos de cómputo móvil están siendo víctimas de un elevado número de ataques en sus comunicaciones debido a su popularidad, estos dispositivos requieren de sistemas criptográficos verdaderamente eficientes que eviten la pérdida de su desempeño debido a lo limitado de sus recursos de procesamiento, siendo la criptografía de curva elíptica quien se las puede brindar.

La implementación en lenguaje java de los sistemas criptográficos estudiados, permitió confirmar que las versiones que operan con algoritmos que emplean curvas elípticas son igualmente de adaptables sobre la misma plataforma, además de que estos puedan ejecutarse indistintamente en cualquier dispositivo que soporte el lenguaje sin importar su tipo de hardware, principalmente en los dispositivos de cómputo móvil que es donde el lenguaje Java tiene mayor auge.

Es necesario señalar que lo anterior solamente se puede lograr escogiendo los parámetros adecuados que se utilizarán con los algoritmos durante la práctica, para ello se deben analizar aspectos como la capacidad de procesamiento de los equipos

actuales, considerar el lenguaje y sistema operativo en el que se implementaran los sistemas, etc.

Por último, cabe destacar que la criptografía de curva elíptica, al igual que los sistemas criptográficos asimétricos utilizados actualmente, basa su seguridad en un problema matemático que con los conocimientos y tecnologías actuales no puede ser resuelto en un tiempo razonable. Sin embargo, la ciencia y la tecnología hoy en día avanzan a pasos agigantados y su solución podría encontrarse muy pronto, por lo que es importante investigar nuevas técnicas de cifrado.

Apéndices

Apéndice A: Algoritmo extendido de Euclides e inversos modulares

El *algoritmo de Euclides* es un método antiguo y eficaz para calcular el *máximo común divisor* de dos números (el mayor número que los divide sin dejar resto). Si a y b son dos números enteros no negativos y $a \geq b$, se define el máximo común divisor de a y b, y denotado como $\text{mcd}(a, b)$, como el entero más grande que divide a a y b.

El algoritmo consiste en varias divisiones sucesivas; en la primera división, se toma como dividendo el mayor de los números y como divisor el segundo número. Luego, el divisor y el resto sirven respectivamente de dividendo y divisor de la siguiente división. El proceso termina cuando se obtiene un resto nulo, entonces el máximo común divisor es el penúltimo resto del algoritmo. Este algoritmo tiene una complejidad $\mathcal{O}(\log n)^2$ para calcular el $\text{mcd}(n, m)$, donde $n \geq m$.

El pseudocódigo para implementar el algoritmo consiste en lo siguiente:

ENTRADA: dos enteros no negativos a y b, $a \geq b$.
SALIDA: $a = \text{mcd}(a, b)$.

1. Mientras $b \neq 0$, hacer lo siguiente:

$$r = a \bmod b$$
$$a = b$$
$$b = r$$

2. Regresar a .

El *algoritmo de Euclides extendido* es una ligera modificación que permite además expresar al máximo común divisor como una combinación lineal. Sean a y b dos números enteros no negativos tales que $a \geq b$ y $d = \text{mcd}(a, b)$, el algoritmo de Euclides se puede extender para encontrar dos números enteros x y y que satisfagan la ecuación $ax + by = d$. El pseudocódigo del algoritmo consiste en lo siguiente:

ENTRADA: dos enteros no negativos a y b, $a \geq b$.
SALIDA: $d = \text{mcd}(a, b)$, x y y que satisfacen la ecuación: $ax + by = d$.

1. Si $b = 0$, hacer lo siguiente: $d = a, x = 1, y = 0$, Regresar d, x, y .
2. Hacer $x_2 = 1, x_1 = 0, y_2 = 0, y_1 = 1$.
3. Mientras $b > 0$, hacer lo siguiente:
   $q = a/b$, $r = a - qb$, $x = x_2 - qx_1$, $y = y_2 - qy_1$, $a = b$, $b = r$, $x_2 = x_1, x_1 = x, y_2 = y_1$ y $y_1 = y$.

4. Hacer $d = a, x = x_2, y = y_2$.
5. Regresar d, x, y .

El algoritmo anterior tiene una complejidad $\mathcal{O}\ \log n\ ^2$ para calcular el mcd(n, m), donde $n \geq m$.

Dos números enteros a y b son congruentes módulo p, si al dividirlos entre p se obtiene el mismo residuo. Cuando a es congruente con $b \bmod p$ se escribe $a \equiv b \bmod p$. Supóngase que se conocen los valores de a, b y p, pero que se desconoce el valor c de la ecuación $ac \equiv b \bmod p$. Basta con encontrar un valor a^{-1} que tenga la característica de que $a^{-1}a \equiv 1 \bmod p$, pues de esta manera se tendría la solución deseada $c \equiv a^{-1}b \bmod p$.

Al valor a^{-1} se le llama *inverso modular* de $a \bmod p$. Desafortunadamente este valor no siempre existe. Este valor existe si y sólo si el máximo común divisor de a y p es 1. Al usar el algoritmo de Euclides extendido se obtiene $px + ay = 1$, el valor y es el inverso modular de $a \bmod p$.

Para la implementación[XXIII] de este algoritmo se emplearon los siguientes parámetros:

- Primer número $a = 23$.
- Segundo número $b = 11$.

La salida del programa fue la siguiente:

```
mcd(23, 11) = 1

Combinación lineal ax + by = d: (23)(1) + (11)(-2) = 1

a = 23, x = 1, b = 11, y = -2, d = 1

El inverso modular de 11 modulo 23 es: 21
```

[XXIII] El código fuente del programa en lenguaje Java se encuentra en el apéndice B.

Apéndice B: Implementación en lenguaje Java del algoritmo extendido de Euclides con cálculo de inverso modular

```
public class Euclides_Extendido {

    public static int d, x, y, invrso;

    public static void main(String args[]){
        int a = 23, b = 11;
        ecldes_extnddo(a, b);
        System.out.printf("mcd(%d, %d) = %d\n", a, b, d);
        System.out.printf("Combinación lineal ax + by = d: (%d)(%d) +
(%d)(%d) = %d\n", a, x, b, y, d);
        System.out.printf("a = %d, x = %d, b = %d, y = %d, d = %d\n", a,
x, b, y, d);
        if(d == 1){
            System.out.printf("El inverso modular de %d modulo %d es:
%d\n", b, a, invrso);
        }

    }

    public static void ecldes_extnddo(int a, int b){
        int x1, x2, y1, y2, q, r, mdlo = a;
        double a_dvddo_por_b;
        if(a >= b){
            if(b == 0){
                d = a;
                x = 1;
                y = 0;
            }
            x2 = 1;
            x1 = 0;
            y2 = 0;
            y1 = 1;
            while(b > 0){
                a_dvddo_por_b = a / b;
                q = (int)Math.floor(a_dvddo_por_b);
                r = a - q * b;
                x = x2 - q * x1;
                y = y2 - q * y1;
                a = b;
                b = r;
                x2 = x1;
                x1 = x;
                y2 = y1;
                y1 = y;
```

```
            }
            d = a;
            x = x2;
            y = y2;
            invrso = y;
            while(invrso < 0){
                invrso = invrso + mdlo;
            }
            while(invrso >= mdlo){
                invrso = invrso - mdlo;
            }
        }
    }

}
```

Apéndice C: Implementación en lenguaje Java del algoritmo para la suma y multiplicación escalar de puntos en una curva elíptica sobre $\mathbb{Z}_p$

```
public class Suma_Puntos_Curva_Eliptica {

    public static int x3, y3, lmbda, kx, ky, A = 3, B = 4, p = 7;
    public static int d, x, y, invrso;

    public static void main(String args[]){
        int x1 = 2, y1 = 2, x2 = 5, y2 = 5, k = 5;
        System.out.printf("Curva elíptica E: y%c2 = x%c3 + %dx + %d mod
%d\n", 94, 94, A, B, p);
        System.out.printf("P1 = (%d, %d)\n", x1, y1);
        System.out.printf("P2 = (%d, %d)\n", x2, y2);
        sma_pntos_crva_elptca(x1, y1, x2, y2);
        System.out.printf("P1 + P2 = (%d, %d)\n", x3, y3);
        mltplccion_esclar(k, x1, y1);
        System.out.printf("Multiplicación escalar kP1 = %dP1 = (%d,
%d)\n", k, kx, ky);
    }

    public static void ecldes_extnddo(int a, int b){
        int x1, x2, y1, y2, q, r, mdlo = a;
        double a_dvddo_por_b;
        if(a >= b){
            if(b == 0){
                d = a;
                x = 1;
                y = 0;
            }
            x2 = 1;
            x1 = 0;
            y2 = 0;
            y1 = 1;
            while(b > 0){
                a_dvddo_por_b = a / b;
                q = (int)Math.floor(a_dvddo_por_b);
                r = a - q * b;
                x = x2 - q * x1;
                y = y2 - q * y1;
                a = b;
                b = r;
                x2 = x1;
                x1 = x;
```

```
                y2 = y1;
                y1 = y;
            }
            d = a;
            x = x2;
            y = y2;
            invrso = y;
            while(invrso < 0){
                invrso = invrso + mdlo;
            }
            while(invrso >= mdlo){
                invrso = invrso - mdlo;
            }
        }
    }

    public static void sma_pntos_crva_elptca(int x1, int y1, int x2, int
y2){
        int nmrdor, dnmndor;
        if(x1 == x2 && y1 == y2 * -1){
            System.out.printf("Error, punto en el infinito\n");
        }
        if(x1 != x2){
            nmrdor = y2 - y1;
            while(nmrdor < 0){
                nmrdor = nmrdor + p;
      64     }
            while(nmrdor >= p){
                nmrdor = nmrdor - p;
            }
            dnmndor = x2 - x1;
            while(dnmndor < 0){
                dnmndor = dnmndor + p;
            }
            while(dnmndor >= p){
                dnmndor = dnmndor - p;
            }
            if(nmrdor % dnmndor == 0){
                lmbda = nmrdor / dnmndor;
            }
            if(nmrdor % dnmndor != 0){
                ecldes_extnddo(p, dnmndor);
                dnmndor = invrso;
                lmbda = (dnmndor * nmrdor) % p;
            }
            x3 = lmbda * lmbda - x1 - x2;
            while(x3 < 0){
                x3 = x3 + p;
            }
            while(x3 >= p){
                x3 = x3 - p;
            }
            y3 = lmbda * (x1 - x3) - y1;
            while(y3 < 0){
                y3 = y3 + p;
            }
            while(y3 >= p){
```

```
            y3 = y3 - p;
        }
    }
    if(x1 == x2 && y1 == y2 && y1 != 0 ){
        nmrdor = 3 * x1 * x1 + A;
        while(nmrdor < 0){
            nmrdor = nmrdor + p;
        }
        while(nmrdor >= p){
            nmrdor = nmrdor - p;
        }
        dnmndor = 2 * y1;
        while(dnmndor < 0){
            dnmndor = dnmndor + p;
        }
        while(dnmndor >= p){
            dnmndor = dnmndor - p;
        }
        if(nmrdor % dnmndor == 0){
            lmbda = nmrdor / dnmndor;
        }
        if(nmrdor % dnmndor != 0){
            ecldes_extnddo(p, dnmndor);
            dnmndor = invrso;
            lmbda = (dnmndor * nmrdor) % p;
        }
        x3 = lmbda * lmbda - 2 * x1;
        while(x3 < 0){
            x3 = x3 + p;
        }
        while(x3 >= p){
            x3 = x3 - p;
        }
        y3 = lmbda * (x1 - x3) - y1;
        while(y3 < 0){
            y3 = y3 + p;
        }
        while(y3 >= p){
            y3 = y3 - p;
        }
    }
}

public static void mltplccion_esclar(int k, int x, int y){
    if(k == 0){
        kx = 0;
        ky = 0;
    }
    if(k == 1){
        kx = x;
        ky = y;
    }
    if(k > 1){
        sma_pntos_crva_elptca(x, y, x, y);
        kx = x3;
        ky = y3;
        for(int i = 2; i < k; i++){
```

```
                sma_pntos_crva_elptca(kx, ky, x, y);
                kx = x3;
                ky = y3;
            }
        }
    }

}
```

Apéndice D: Implementación en lenguaje Java del algoritmo Diffie – Hellman para el intercambio de claves

```
import java.util.Random;

public class Diffie_Hellman {

    public static long alfa = 5, p = 11; //EL GRUPO "G" MULTIPLICATIVO
FINITO ES Z_11

    public static void main(String args[]){
        diffie_hellman();
    }

    public static long gnra_nmero_altrio_prvado(){
        long x;
        Random r = new Random();
        x = r.nextLong() % p; //SE ASEGURA QUE EL NÚMERO ALEATORIO
GENERADO PERTENEZCA A G.
        if (x == 0) x = 1; //SE ASEGURA QUE EL NÚMERO ALEATORIO GENERADO
PERTENEZCA A G.
        if (x < 0 ) x = x * -1; //SE ASEGURA QUE EL NÚMERO ALEATORIO
GENERADO PERTENEZCA A G.
        return x;
    }

    public static long a_elvdo_b_mod_p(long a, long b){
         long resltdo = a;
         for(int i = 1; i < b; i++){
             resltdo = resltdo * a;
         }
         return resltdo % p;
     }

    public static void diffie_hellman(){
        long a, b, a_pblico, b_pblico, nmro_scrto_A, nmro_scrto_B;
        a = gnra_nmero_altrio_prvado(); //GENERA EL NÚMERO ALEATORIO DEL
USUARIO "A" QUE PERTENECE AL GRUPO "G"
        a_pblico = a_elvdo_b_mod_p(alfa, a);
        System.out.print("Número público transmitido por el usuario A al
usuario B: " + a_pblico + "\n");
        b = gnra_nmero_altrio_prvado(); //GENERA EL NÚMERO ALEATORIO DEL
USUARIO "B" QUE PERTENECE AL GRUPO "G"
        b_pblico = a_elvdo_b_mod_p(alfa, b);
        System.out.print("Número público transmitido por el usuario B al
usuario A: " + b_pblico + "\n");
        nmro_scrto_A = a_elvdo_b_mod_p(b_pblico, a);
```

```
        System.out.print("Número secreto calculado por el usuario A: " +
nmro_scrto_A + "\n");
        nmro_scrto_B = a_elvdo_b_mod_p(a_pblico, b);
        System.out.print("Número secreto calculado por el usuario B: " +
nmro_scrto_B + "\n");
    }

}
```

Apéndice E: Implementación en lenguaje Java del sistema criptográfico asimétrico ElGamal

```
import java.io.*;
import java.util.Locale;
import java.util.Scanner;

public class El_Gamal {

    public static int d, x, y, invrso;
    public static int p = 127, g = 2, b_A, g_elvdo_k_mod_p,
M_mltplcdo_b_A_elvdo_k_mod_p;

    public static void main(String args[]){
        el_gamal();
    }

    public static void ecldes_extnddo(int a, int b){
        int x1, x2, y1, y2, q, r, mdlo = a;
        double a_dvddo_por_b;
        if(a >= b){
            if(b == 0){
                d = a;
                x = 1;
                y = 0;
            }
            x2 = 1;
            x1 = 0;
            y2 = 0;
            y1 = 1;
            while(b > 0){
                a_dvddo_por_b = a / b;
                q = (int)Math.floor(a_dvddo_por_b);
                r = a - q * b;
                x = x2 - q * x1;
                y = y2 - q * y1;
                a = b;
                b = r;
                x2 = x1;
                x1 = x;
                y2 = y1;
                y1 = y;
            }
            d = a;
            x = x2;
            y = y2;
            invrso = y;
```

```
            while(invrso < 0){
                invrso = invrso + mdlo;
            }
            while(invrso >= mdlo){
                invrso = invrso - mdlo;
            }
        }
    }

    public static int a_elvdo_b_mod_p(int a, int b){
         int resltdo = a;
         for(int i = 1; i < b; i++){
             resltdo = resltdo * a;
         }
         return resltdo % p;
     }

    public static String lee_cdna_tcldo(){
        String cdna_lda = "";
        try{
            InputStreamReader inptstrmrder = new
InputStreamReader(System.in);
            BufferedReader fljo_entrda = new
BufferedReader(inptstrmrder);
            cdna_lda = fljo_entrda.readLine();
        }
        catch(IOException error){
            System.err.println("Error: " + error.getMessage());
        }
        return cdna_lda;
    }

    public static void verifica_cdna(String cdna_lda){
         int vlor_encntrdo;
         for(int i = 0; i < cdna_lda.length(); i++){
             vlor_encntrdo = cdna_lda.charAt(i);
             if(vlor_encntrdo < 32 || vlor_encntrdo > 126){ //VERIFICA
QUE LA CADENA LEIDA SOLO CONTENGA CARACTERES DEL ASCII IMPRIMIBLE
(EQUIVALENTE DECIMAL DEL 32 AL 126).
                 System.out.printf("Eror: solo se aceptan caracteres del
ASCII imprimible\n");
                 System.exit(0);
             }
         }
     }

    public static void cfra_cdna(String cdna_lda){
         int M, k, b_A_elvdo_k_mod_p;
         for(int i = 0; i < cdna_lda.length(); i++){
             M = cdna_lda.charAt(i);
             k = (int)(Math.random()*10+1); //EL VALOR ALEATORIO DE K SE
TRUNCA A 10 DEBIDOA QUE EL TIPO DE DATO INT SOLO ES DE 32 BITS.
             g_elvdo_k_mod_p = a_elvdo_b_mod_p(g, k);
             b_A_elvdo_k_mod_p = a_elvdo_b_mod_p(b_A, k);
             M_mltplcdo_b_A_elvdo_k_mod_p = (M * b_A_elvdo_k_mod_p) % p;
             System.out.printf("%d %d ",g_elvdo_k_mod_p,
M_mltplcdo_b_A_elvdo_k_mod_p);
```

```
            }
        }

    public static void dscfra_cdna(String cdna, int a_A){
        int M, g_elvdo_k_elvdo_a_A_mod_p,
invrso_g_elvdo_k_elvdo_a_A_mod_p;
        long D_B;
        Scanner leer = new Scanner(cdna);
        leer.useLocale(new Locale("en", "US"));
        while(leer.hasNext()){
            g_elvdo_k_mod_p = leer.nextInt();
            M_mltplcdo_b_A_elvdo_k_mod_p = leer.nextInt();
            g_elvdo_k_elvdo_a_A_mod_p = a_elvdo_b_mod_p(g_elvdo_k_mod_p,
a_A);
            ecldes_extnddo(p, g_elvdo_k_elvdo_a_A_mod_p);
            invrso_g_elvdo_k_elvdo_a_A_mod_p = invrso;
            M = (invrso_g_elvdo_k_elvdo_a_A_mod_p *
M_mltplcdo_b_A_elvdo_k_mod_p) % p;
            System.out. printf("%c", M);
        }
    }

    public static void el_gamal(){
        int a_A = 3, k, g_elvdo_a_A_elvdo_k_mod_p;
        b_A = a_elvdo_b_mod_p(g, a_A);
        System.out.print("Ingrese cadena: ");
        String cdna_lda = lee_cdna_tcldo();
        verifica_cdna(cdna_lda);
        System.out.print("Cadena cifrada: ");
        cfra_cdna(cdna_lda);
        System.out.print("\nIngrese cadena a descifrar: ");
        String cdna_lda2 = lee_cdna_tcldo();
        System.out.print("Cadena descifrada: ");
        dscfra_cdna(cdna_lda2, a_A);
        System.out.print("\n");
    }

}
```

Apéndice F: Implementación en lenguaje Java del sistema criptográfico asimétrico Massey – Omura

```
import java.io.*;
import java.util.Locale;
import java.util.Scanner;

public class Masey_Omura {

    public static long e_A = 9, d_A = 9, e_B = 11, d_B = 11, p = 41, M,
E_A, E_B, D_A, D_B;
    public static char[] alfbto = new char[27];

    public static void main(String args[]){
        masey_omura();
    }

    public static void llna_alfbto(){
        for(int i = 0; i < alfbto.length - 1; i++){
            alfbto[i] = (char)(i + 65);
        }
        alfbto[alfbto.length - 1] = (char)(' ');
    }

    public static String lee_cdna_tcldo(){
        String cdna_lda = "";
        try{
            InputStreamReader inptstrmrder = new
InputStreamReader(System.in);
            BufferedReader fljo_entrda = new
BufferedReader(inptstrmrder);
            cdna_lda = fljo_entrda.readLine();
        }
        catch(IOException error){
            System.err.println("Error: " + error.getMessage());
        }
        return cdna_lda.toUpperCase();
    }

     public static void cfra_cdna(String cdna_lda){
         char crcter;
         for(int i = 0; i < cdna_lda.length(); i++){
             crcter = cdna_lda.charAt(i);
             for(int j = 0; j < alfbto.length; j++){
                 if(crcter == alfbto[j]){
                     E_A = a_elvdo_b_mod_p(j + 1,e_A); //SE SUMA 1 PORQUE
EL ARREGLO COMIENZA EN EL ELEMENTO 1.
```

```
                    E_B = a_elvdo_b_mod_p(E_A,e_B);
                    D_A = a_elvdo_b_mod_p(E_B,d_A);
                    System.out.print(D_A + " ");
                }
            }
        }
    }

    public static void dscfra_cdna(String cdna){
        int vlor_encntrdo;
        long D_B;
        Scanner leer = new Scanner(cdna);
        leer.useLocale(new Locale("en", "US"));
        while(leer.hasNext()){
            vlor_encntrdo = leer.nextInt();
            D_B = (long) a_elvdo_b_mod_p(vlor_encntrdo,d_B);
            System.out.print(alfbto[(int)(D_B - 1)]); //SE RESTA 1
PORQUE EL ARREGLO COMIENZA EN EL ELEMENTO 0.
        }
    }

    public static long a_elvdo_b_mod_p(long a, long b){
        long resltdo = a;
        for(int i = 1; i < b; i++){
            resltdo = resltdo * a;
        }
        return resltdo % p;
    }

    public static void masey_omura(){
       llna_alfbto();
       System.out.print("Ingrese cadena: ");
       String cdna_lda = lee_cdna_tcldo();
       System.out.print("Cadena cifrada: ");
       cfra_cdna(cdna_lda);
       System.out.print("\nIngrese cadena a descifrar: ");
       String cdna_lda2 = lee_cdna_tcldo();
       System.out.print("Cadena descifrada: ");
       dscfra_cdna(cdna_lda2);
       System.out.print("\n");
    }

}
```

Apéndice G: Implementación en lenguaje Java del algoritmo Diffie – Hellman para el intercambio de claves empleando curvas elípticas

```
public class Diffie_Hellman_Curvas_Elipticas {

    public static int x3, y3, lmbda, kx, ky, A = 1, B = 1, p = 23;
    public static int d, x, y, invrso;
    public static int x_pblco = 3, y_pblco = 10, a = 15, b = 19,
aG_x_pblco, aG_y_pblco, bG_x_pblco, bG_y_pblco;

    public static void main(String args[]){
        diffie_hellman_curvas_elipticas();
    }

    public static void ecldes_extnddo(int a, int b){
        int x1, x2, y1, y2, q, r, mdlo = a;
        double a_dvddo_por_b;
        if(a >= b){
            if(b == 0){
                d = a;
                x = 1;
                y = 0;
            }
            x2 = 1;
            x1 = 0;
            y2 = 0;
            y1 = 1;
            while(b > 0){
                a_dvddo_por_b = a / b;
                q = (int)Math.floor(a_dvddo_por_b);
                r = a - q * b;
                x = x2 - q * x1;
                y = y2 - q * y1;
                a = b;
                b = r;
                x2 = x1;
                x1 = x;
                y2 = y1;
                y1 = y;
            }
            d = a;
            x = x2;
            y = y2;
            invrso = y;
```

```
            while(invrso < 0){
                invrso = invrso + mdlo;
            }
            while(invrso >= mdlo){
                invrso = invrso - mdlo;
            }
        }
    }

    public static void sma_pntos_crva_elptca(int x1, int y1, int x2, int
y2){
        int nmrdor, dnmndor;
        if(x1 == x2 && y1 == y2 * -1){
            System.out.printf("Error, punto en el infinito\n");
        }
        if(x1 != x2){
            nmrdor = y2 - y1;
            while(nmrdor < 0){
                nmrdor = nmrdor + p;
            }
            while(nmrdor >= p){
                nmrdor = nmrdor - p;
            }
            dnmndor = x2 - x1;
            while(dnmndor < 0){
                dnmndor = dnmndor + p;
            }
            while(dnmndor >= p){
                dnmndor = dnmndor - p;
            }
            if(nmrdor % dnmndor == 0){
                lmbda = nmrdor / dnmndor;
            }
            if(nmrdor % dnmndor != 0){
                ecldes_extnddo(p, dnmndor);
                dnmndor = invrso;
                lmbda = (dnmndor * nmrdor) % p;
            }
            x3 = lmbda * lmbda - x1 - x2;
            while(x3 < 0){
                x3 = x3 + p;
            }
            while(x3 >= p){
                x3 = x3 - p;
            }
            y3 = lmbda * (x1 - x3) - y1;
            while(y3 < 0){
                y3 = y3 + p;
            }
            while(y3 >= p){
                y3 = y3 - p;
            }
        }
        if(x1 == x2 && y1 == y2 && y1 != 0 ){
            nmrdor = 3 * x1 * x1 + A;
            while(nmrdor < 0){
                nmrdor = nmrdor + p;
```

```
        }
        while(nmrdor >= p){
            nmrdor = nmrdor - p;
        }
        dnmndor = 2 * y1;
        while(dnmndor < 0){
            dnmndor = dnmndor + p;
        }
        while(dnmndor >= p){
            dnmndor = dnmndor - p;
        }
        if(nmrdor % dnmndor == 0){
            lmbda = nmrdor / dnmndor;
        }
        if(nmrdor % dnmndor != 0){
            ecldes_extnddo(p, dnmndor);
            dnmndor = invrso;
            lmbda = (dnmndor * nmrdor) % p;
        }
        x3 = lmbda * lmbda - 2 * x1;
        while(x3 < 0){
            x3 = x3 + p;
        }
        while(x3 >= p){
            x3 = x3 - p;
        }
        y3 = lmbda * (x1 - x3) - y1;
        while(y3 < 0){
            y3 = y3 + p;
        }
        while(y3 >= p){
            y3 = y3 - p;
        }
    }
}

public static void mltplccion_esclar(int k, int x, int y){
    if(k == 0){
        kx = 0;
        ky = 0;
    }
    if(k == 1){
        kx = x;
        ky = y;
    }
    if(k > 1){
        sma_pntos_crva_elptca(x, y, x, y);
        kx = x3;
        ky = y3;
        for(int i = 2; i < k; i++){
            sma_pntos_crva_elptca(kx, ky, x, y);
            kx = x3;
            ky = y3;
        }
    }
}
```

```
    public static void diffie_hellman_curvas_elipticas(){
        int abG_x_scrto, abG_y_scrto, baG_x_scrto, baG_y_scrto;
        mltplccion_esclar(a, x_pblco, y_pblco);
        aG_x_pblco = kx;
        aG_y_pblco = ky;
        System.out.printf("Punto público transmitido por el usuario A al
usuario B: (%d, %d)\n",aG_x_pblco, aG_y_pblco);
        mltplccion_esclar(b, x_pblco, y_pblco);
        bG_x_pblco = kx;
        bG_y_pblco = ky;
        System.out.printf("Punto público transmitido por el usuario B al
usuario A: (%d, %d)\n",bG_x_pblco, bG_y_pblco);
        mltplccion_esclar(a, bG_x_pblco, bG_y_pblco);
        abG_x_scrto = kx;
        abG_y_scrto = ky;
        System.out.printf("Punto secreto calculado por el usuario A: (%d,
%d)\n", abG_x_scrto, abG_y_scrto);
        mltplccion_esclar(b, aG_x_pblco, aG_y_pblco);
        baG_x_scrto = kx;
        baG_y_scrto = ky;
        System.out.printf("Punto secreto calculado por el usuario B: (%d,
%d)\n", baG_x_scrto, baG_y_scrto);
    }

}
```

Apéndice H: Implementación en lenguaje Java del sistema criptográfico asimétrico ElGamal empleando curvas elípticas

```
import java.io.*;
import java.util.Locale;
import java.util.Scanner;
import java.util.Random;


public class el_gamal_curva_eliptica {

    public static int x3, y3, lmbda, kx, ky, A = 1, B = 2, p = 97;
    public static int d, x, y, invrso;
    public static int P0_x = 41, P0_y = 71, a_A = 2;
    public static int[] alfbto = new int[95];
    public static int[][] alfbto_cfrdo = new int[103][2];

    public static void main(String args[]){
        el_gamal_curva_eliptica();
    }

    public static void dscfra_cdna(String cdna){
        int crcter_ldo, kP0_x, kP0_y, kPA_P_x, kPA_P_y, kPA_x_P, kPA_y_P,
kPA_x, kPA_y, ngtvo_kPA_y_P = 0, P_x, P_y;
        Scanner leer = new Scanner(cdna);
        leer.useLocale(new Locale("en", "US"));
        while(leer.hasNext()){
            kP0_x = leer.nextInt();
            kP0_y = leer.nextInt();
            kPA_P_x = leer.nextInt();
            kPA_P_y = leer.nextInt();
            mltplccion_esclar(a_A, kP0_x, kP0_y);
            kPA_x_P = kx;
            kPA_y_P = ky;
            for(int i = 0; i < alfbto_cfrdo.length; i++){
                if(kPA_x_P == alfbto_cfrdo[i][0] && kPA_y_P !=
alfbto_cfrdo[i][1]){
                    ngtvo_kPA_y_P = alfbto_cfrdo[i][1];
                }
            }
            sma_pntos_crva_elptca(kPA_P_x, kPA_P_y, kPA_x_P,
ngtvo_kPA_y_P);
            P_x = x3;
            P_y = y3;
            for(int j = 0; j < alfbto.length; j++){
                if(P_x == alfbto_cfrdo[j][0] && P_y ==
alfbto_cfrdo[j][1]){
```

```
                    System.out.printf("%c", alfbto[j]);
                }
            }
        }
    }

    public static void cfra_cdna(String cdna_lda){
        int crcter_ldo, k = 0, kP0_x, kP0_y, ka_A, kPA_x, kPA_y, kPA_x_P,
kPA_y_P;
        for(int i = 0; i < cdna_lda.length(); i++){
            crcter_ldo = (int)cdna_lda.charAt(i);
            for(int j = 0; j < alfbto.length; j++){
                if(crcter_ldo == alfbto[j]){
                    k = gnra_nmero_altrio();
                    mltplccion_esclar(k, P0_x, P0_y);
                    kP0_x = kx;
                    kP0_y = ky;
                    ka_A = k * a_A;
                    mltplccion_esclar(ka_A, P0_x, P0_y);
                    kPA_x = kx;
                    kPA_y = ky;
                    sma_pntos_crva_elptca(kPA_x, kPA_y,
alfbto_cfrdo[j][0], alfbto_cfrdo[j][1]);
                    kPA_x_P = x3;
                    kPA_y_P = y3;
                    System.out.printf("%d %d %d %d ",kP0_x, kP0_y,
kPA_x_P, kPA_y_P);
                }
            }
        }
    }

    public static void cfra_alfbto(){
        int izqrda, drcha, indce_alfbto_cfrdo = 0;
        for(int i = 0; i < alfbto.length; i++){
            alfbto[i] = i + 32;
        }
        for(int x = 0; x < p; x++){
            drcha = ((int)(Math.pow(x, 3)) + A * x + B) % p;
            for(int y = 0; y < p ; y++){
                izqrda = ((int)Math.pow(y, 2)) % p;
                if(drcha == izqrda){
                    alfbto_cfrdo[indce_alfbto_cfrdo][0] = x;
                    alfbto_cfrdo[indce_alfbto_cfrdo][1] = y;
                    indce_alfbto_cfrdo++;
                }
            }
        }
    }

    public static int gnra_nmero_altrio(){
        int x;
        Random r = new Random();
        x = r.nextInt() % 50;
        if (x == 0) x = 1;
        if (x < 0 ) x = x * -1;
        return x;
```

```
    }

    public static String lee_cdna_tcldo(){
        String cdna_lda = "";
        try{
            InputStreamReader inptstrmrder = new
InputStreamReader(System.in);
            BufferedReader fljo_entrda = new
BufferedReader(inptstrmrder);
            cdna_lda = fljo_entrda.readLine();
        }
        catch(IOException error){
            System.err.println("Error: " + error.getMessage());
        }
        return cdna_lda;
    }

    public static void el_gamal_curva_eliptica(){
        cfra_alfbto();
        System.out.print("Ingrese cadena: ");
        String cdna_lda = lee_cdna_tcldo();
        System.out.print("Cadena cifrada: ");
        cfra_cdna(cdna_lda);
        System.out.print("\nIngrese cadena a descifrar: ");
        String cdna_lda2 = lee_cdna_tcldo();
        System.out.print("Cadena descifrada: ");
        dscfra_cdna(cdna_lda2);
        System.out.print("\n");
    }

    public static void ecldes_extnddo(int a, int b){
        int x1, x2, y1, y2, q, r, mdlo = a;
        double a_dvddo_por_b;
        if(a >= b){
            if(b == 0){
                d = a;
                x = 1;
                y = 0;
            }
            x2 = 1;
            x1 = 0;
            y2 = 0;
            y1 = 1;
            while(b > 0){
                a_dvddo_por_b = a / b;
                q = (int)Math.floor(a_dvddo_por_b);
                r = a - q * b;
                x = x2 - q * x1;
                y = y2 - q * y1;
                a = b;
                b = r;
                x2 = x1;
                x1 = x;
                y2 = y1;
                y1 = y;
            }
            d = a;
```

```
            x = x2;
            y = y2;
            invrso = y;
            while(invrso < 0){
                invrso = invrso + mdlo;
            }
            while(invrso >= mdlo){
                invrso = invrso - mdlo;
            }
        }
    }

    public static void sma_pntos_crva_elptca(int x1, int y1, int x2, int
y2){
        int nmrdor, dnmndor;
        if(x1 == x2 && y1 == y2 * -1){
            System.out.printf("Error, punto en el infinito\n");
        }
        if(x1 != x2){
            nmrdor = y2 - y1;
            while(nmrdor < 0){
                nmrdor = nmrdor + p;
            }
            while(nmrdor >= p){
                nmrdor = nmrdor - p;
            }
            dnmndor = x2 - x1;
            while(dnmndor < 0){
                dnmndor = dnmndor + p;
            }
            while(dnmndor >= p){
                dnmndor = dnmndor - p;
            }
            if(nmrdor % dnmndor == 0){
                lmbda = nmrdor / dnmndor;
            }
            if(nmrdor % dnmndor != 0){
                ecldes_extnddo(p, dnmndor);
                dnmndor = invrso;
                lmbda = (dnmndor * nmrdor) % p;
            }
            x3 = lmbda * lmbda - x1 - x2;
            while(x3 < 0){
                x3 = x3 + p;
            }
            while(x3 >= p){
                x3 = x3 - p;
            }
            y3 = lmbda * (x1 - x3) - y1;
            while(y3 < 0){
                y3 = y3 + p;
            }
            while(y3 >= p){
                y3 = y3 - p;
            }
        }
        if(x1 == x2 && y1 == y2 && y1 != 0 ){
```

```
        nmrdor = 3 * x1 * x1 + A;
        while(nmrdor < 0){
            nmrdor = nmrdor + p;
        }
        while(nmrdor >= p){
            nmrdor = nmrdor - p;
        }
        dnmndor = 2 * y1;
        while(dnmndor < 0){
            dnmndor = dnmndor + p;
        }
        while(dnmndor >= p){
            dnmndor = dnmndor - p;
        }
        if(nmrdor % dnmndor == 0){
            lmbda = nmrdor / dnmndor;
        }
        if(nmrdor % dnmndor != 0){
            ecldes_extnddo(p, dnmndor);
            dnmndor = invrso;
            lmbda = (dnmndor * nmrdor) % p;
        }
        x3 = lmbda * lmbda - 2 * x1;
        while(x3 < 0){
            x3 = x3 + p;
        }
        while(x3 >= p){
            x3 = x3 - p;
        }
        y3 = lmbda * (x1 - x3) - y1;
        while(y3 < 0){
            y3 = y3 + p;
        }
        while(y3 >= p){
            y3 = y3 - p;
        }
    }
}

public static void mltplccion_esclar(int k, int x, int y){
    if(k == 0){
        kx = 0;
        ky = 0;
    }
    if(k == 1){
        kx = x;
        ky = y;
    }
    if(k > 1){
        sma_pntos_crva_elptca(x, y, x, y);
        kx = x3;
        ky = y3;
        for(int i = 2; i < k; i++){
            sma_pntos_crva_elptca(kx, ky, x, y);
            kx = x3;
            ky = y3;
        }
```

```
            }
        }

}
```

Apéndice I: Implementación en lenguaje Java del sistema criptográfico asimétrico Massey – Omura empleando curvas elípticas

```
import java.io.*;
import java.util.Locale;
import java.util.Scanner;

public class massey_omura_curva_eliptica {

    public static int x3, y3, lmbda, kx, ky, A = 3, B = 4, p = 103;
    public static int d, x, y, invrso;
    public static int e_A = 5, d_A = 23, e_B = 7, d_B = 49;
    public static int[] alfbto = new int[95];
    public static int[][] alfbto_cfrdo = new int[114][2];

    public static void main(String args[]){
        massey_omura_curva_eliptica();
    }

    public static void dscfra_cdna(String cdna){
        int crcter_ldo, P_x, P_y;
        Scanner leer = new Scanner(cdna);
        leer.useLocale(new Locale("en", "US"));
        while(leer.hasNext()){
            P_x = leer.nextInt();
            P_y = leer.nextInt();
            mltplccion_esclar(d_B, P_x, P_y);
            P_x = x3;
            P_y = y3;
            for(int i = 0; i < alfbto.length; i++){
                if(P_x == alfbto_cfrdo[i][0] && P_y ==
alfbto_cfrdo[i][1]){
                    System.out.printf("%c", alfbto[i]);
                }
            }
        }
    }

    public static void cfra_cdna(String cdna_lda){
        int crcter_ldo, P_x, P_y;
        for(int i = 0; i < cdna_lda.length(); i++){
            crcter_ldo = (int)cdna_lda.charAt(i);
            for(int j = 0; j < alfbto.length; j++){
                if(crcter_ldo == alfbto[j]){
```

```
                    P_x = alfbto_cfrdo[j][0];
                    P_y = alfbto_cfrdo[j][1];
                    mltplccion_esclar(e_A, P_x, P_y);
                    P_x = x3;
                    P_y = y3;
                    mltplccion_esclar(e_B, P_x, P_y);
                    P_x = x3;
                    P_y = y3;
                    mltplccion_esclar(d_A, P_x, P_y);
                    P_x = x3;
                    P_y = y3;
                    System.out.printf("%d %d ", P_x, P_y);
                }
            }
        }
    }

    public static void cfra_alfbto(){
        int izqrda, drcha, indce_alfbto_cfrdo = 0;
        for(int i = 0; i < alfbto.length; i++){
            alfbto[i] = i + 32;
        }
        for(int x = 0; x < p; x++){
            drcha = ((int)(Math.pow(x, 3)) + A * x + B) % p;
            for(int y = 0; y < p ; y++){
                izqrda = ((int)Math.pow(y, 2)) % p;
                if(drcha == izqrda){
                    alfbto_cfrdo[indce_alfbto_cfrdo][0] = x;
                    alfbto_cfrdo[indce_alfbto_cfrdo][1] = y;
                    indce_alfbto_cfrdo++;
                }
            }
        }
    }

    public static String lee_cdna_tcldo(){
        String cdna_lda = "";
        try{
            InputStreamReader inptstrmrder = new
InputStreamReader(System.in);
            BufferedReader fljo_entrda = new
BufferedReader(inptstrmrder);
            cdna_lda = fljo_entrda.readLine();
        }
        catch(IOException error){
            System.err.println("Error: " + error.getMessage());
        }
        return cdna_lda;
    }

    public static void massey_omura_curva_eliptica(){
        cfra_alfbto();
        System.out.print("Ingrese cadena: ");
        String cdna_lda = lee_cdna_tcldo();
        System.out.print("Cadena cifrada: ");
        cfra_cdna(cdna_lda);
        System.out.print("\nIngrese cadena a descifrar: ");
```

```
        String cdna_lda2 = lee_cdna_tcldo();
        System.out.print("Cadena descifrada: ");
        dscfra_cdna(cdna_lda2);
        System.out.print("\n");
    }

    public static void ecldes_extnddo(int a, int b){
        int x1, x2, y1, y2, q, r, mdlo = a;
        double a_dvddo_por_b;
        if(a >= b){
            if(b == 0){
                d = a;
                x = 1;
                y = 0;
            }
            x2 = 1;
            x1 = 0;
            y2 = 0;
            y1 = 1;
            while(b > 0){
                a_dvddo_por_b = a / b;
                q = (int)Math.floor(a_dvddo_por_b);
                r = a - q * b;
                x = x2 - q * x1;
                y = y2 - q * y1;
                a = b;
                b = r;
                x2 = x1;
                x1 = x;
                y2 = y1;
                y1 = y;
            }
            d = a;
            x = x2;
            y = y2;
            invrso = y;
            while(invrso < 0){
                invrso = invrso + mdlo;
            }
            while(invrso >= mdlo){
                invrso = invrso - mdlo;
            }
        }
    }

    public static void sma_pntos_crva_elptca(int x1, int y1, int x2, int
y2){
        int nmrdor, dnmndor;
        if(x1 == x2 && y1 == y2 * -1){
            System.out.printf("Error, punto en el infinito\n");
        }
        if(x1 != x2){
            nmrdor = y2 - y1;
            while(nmrdor < 0){
                nmrdor = nmrdor + p;
            }
            while(nmrdor >= p){
```

```
            nmrdor = nmrdor - p;
        }
        dnmndor = x2 - x1;
        while(dnmndor < 0){
            dnmndor = dnmndor + p;
        }
        while(dnmndor >= p){
            dnmndor = dnmndor - p;
        }
        if(nmrdor % dnmndor == 0){
            lmbda = nmrdor / dnmndor;
        }
        if(nmrdor % dnmndor != 0){
            ecldes_extnddo(p, dnmndor);
            dnmndor = invrso;
            lmbda = (dnmndor * nmrdor) % p;
        }
        x3 = lmbda * lmbda - x1 - x2;
        while(x3 < 0){
            x3 = x3 + p;
        }
        while(x3 >= p){
            x3 = x3 - p;
        }
        y3 = lmbda * (x1 - x3) - y1;
        while(y3 < 0){
            y3 = y3 + p;
        }
        while(y3 >= p){
            y3 = y3 - p;
        }
    }
    if(x1 == x2 && y1 == y2 && y1 != 0 ){
        nmrdor = 3 * x1 * x1 + A;
        while(nmrdor < 0){
            nmrdor = nmrdor + p;
        }
        while(nmrdor >= p){
            nmrdor = nmrdor - p;
        }
        dnmndor = 2 * y1;
        while(dnmndor < 0){
            dnmndor = dnmndor + p;
        }
        while(dnmndor >= p){
            dnmndor = dnmndor - p;
        }
        if(nmrdor % dnmndor == 0){
            lmbda = nmrdor / dnmndor;
        }
        if(nmrdor % dnmndor != 0){
            ecldes_extnddo(p, dnmndor);
            dnmndor = invrso;
            lmbda = (dnmndor * nmrdor) % p;
        }
        x3 = lmbda * lmbda - 2 * x1;
        while(x3 < 0){
```

```
                x3 = x3 + p;
            }
            while(x3 >= p){
                x3 = x3 - p;
            }
            y3 = lmbda * (x1 - x3) - y1;
            while(y3 < 0){
                y3 = y3 + p;
            }
            while(y3 >= p){
                y3 = y3 - p;
            }
        }
    }

    public static void mltplccion_esclar(int k, int x, int y){
        if(k == 0){
            kx = 0;
            ky = 0;
        }
        if(k == 1){
            kx = x;
            ky = y;
        }
        if(k > 1){
            sma_pntos_crva_elptca(x, y, x, y);
            kx = x3;
            ky = y3;
            for(int i = 2; i < k; i++){
                sma_pntos_crva_elptca(kx, ky, x, y);
                kx = x3;
                ky = y3;
            }
        }
    }

}
```

Apéndice J: Glosario

a. C. "antes de Cristo", se emplea para referirse y fechar los años y siglos anteriores a la era cristiana, que comienza con el año convencional del nacimiento de Jesucristo.

Algoritmo de cifrado. Procedimiento que utilizando cierta clave, transforma un texto plano en texto cifrado de tal forma que es incomprensible, o al menos, difícil de comprender.

Algoritmo de cifrado de sustitución. Algoritmo de cifrado creado por el reemplazo de letras o grandes bloques del texto original con sustitutos, usualmente de la misma longitud.

Algoritmo de cifrado de transposición. Algoritmo de cifrado creado utilizando los mismos caracteres y las mismas frecuencias que las del texto original, solo que estas son reordenadas.

Algoritmo de cifrado producto. Algoritmo de cifrado creado por la composición de varios cifrados, cuyos tipos alternan entre sustitución y trasposición.

Algoritmo de descifrado. Procedimiento que utilizando cierta clave, transforma un texto cifrado en texto plano de tal forma que es comprensible.

Ataque sobre un algoritmo de cifrado. Método por el cual un individuo intenta encontrar debilidades en un algoritmo de cifrado para comprometer su seguridad sin el conocimiento de información secreta.

Ataque de solo texto cifrado. Ataque sobre un algoritmo de cifrado en el que solo se conoce el texto cifrado, aunque a menudo el lenguaje del texto plano y el tipo de algoritmo de cifrado también son conocidos. El objetivo del criptoanalista es encontrar el texto plano y la clave.

Ataque de texto plano conocido. Ataque sobre un algoritmo de cifrado en el que el criptoanalista tiene algún texto cifrado y su correspondiente texto plano. El objetivo es encontrar la clave con la que otro texto cifrado puede ser descifrado.

Ataque de texto plano escogido. Ataque sobre un algoritmo de cifrado en el que el criptoanalista puede especificar algún texto plano a cifrar, tal vez un texto sin significado, y de alguna manera aprender como es el correspondiente texto cifrado.

Clave. Pieza de información que controla la operación de un algoritmo de cifrado o de descifrado.

Complejidad polinomial. Tiempo de ejecución de un algoritmo, que es menor que cierto valor calculado usando una fórmula polinomial a partir del número de variables implicadas.

Complejidad temporal de un problema. Número de operaciones que toma resolver el problema a partir del tamaño de la entrada.

Completitud. Cualidad de completo.

Conjunto. Agrupación de objetos simples en un todo. Un conjunto es un medio por el cual se puede hablar de colecciones de objetos de manera abstracta.

Cota superior asintótica. Función que sirve de cota superior de otra función cuando el argumento tiende a infinito. Usualmente se utiliza la notación $\mathcal{O}$ de Landau, coloquialmente llamada "notación de la o grande", para referirse a las funciones acotadas superiormente por otra función.

Criptoanálisis. Estudio de los sistemas criptográficos con el fin de encontrar debilidades en dichos sistemas y romper su seguridad sin el conocimiento de información secreta.

Criptoanalista. Persona que se dedica al criptoanálisis.

Criptografía. Estudio de los algoritmos, protocolos y sistemas que se utilizan para dotar de seguridad a las comunicaciones, a la información y a las entidades que se comunican.

Curva elíptica. Curva plana definida mediante una ecuación de tercer grado, en la que se puede definir una operación binaria para el conjunto de sus puntos y que constituye una estructura de grupo abeliano. El término curva elíptica es un nombre inapropiado ya que las curvas elípticas, no son elipses; el término proviene del hecho de que las curvas elípticas hicieron su aparición inicial durante los primeros intentos para calcular la longitud de arco de una elipse.

Función unidireccional tramposa. Función que puede ser invertida en un espacio de tiempo tolerable cuando se conoce alguna información adicional extra.

Grupo. Estructura algebraica que consta de un conjunto y una operación que combina cualquier pareja de sus elementos para formar un tercer elemento. Cumple con las propiedades asociativa, de cerradura, elemento idéntico y elemento inverso.

Grupo abeliano. Estructura algebraica de grupo que además cumple con la propiedad conmutativa.

Mecanismo de seguridad. Entidad física o lógica, responsable de suministrar un servicio de seguridad.

Operación binaria. Operación matemática, que necesita un operador y dos operandos para que se pueda calcular un valor.

Orden de un grupo. Su cardinalidad, es decir, el número de elementos que constituyen el grupo.

Punto en el infinito. Elemento identidad del grupo de una curva elíptica.

Sistema criptográfico asimétrico o de clave pública. Sistema criptográfico en el que los algoritmos de cifrado y descifrado utilizan claves distintas.

Sistema criptográfico de curva elíptica. Sistema criptográfico asimétrico que en vez de operar con números enteros, opera con coordenadas de puntos de una estructura algebraica llamada curva elíptica.

Sistema criptográfico simétrico o de clave privada. Sistema criptográfico en el que los algoritmos de cifrado y descifrado utilizan la misma clave.

Teoría de la complejidad computacional. Rama de la teoría de la computación, que estudia de manera teórica, la complejidad inherente a la resolución de un problema computable.

Teoría de la computación. Rama de las matemáticas y la computación, que centra su estudio en las limitaciones y capacidades fundamentales de las computadoras.

Texto cifrado. Texto plano que es transformado de tal forma que es incomprensible, o al menos, difícil de comprender.

Texto plano. Texto comprensible.

Tiempo de ejecución de un algoritmo sobre una entrada en particular. Número de operaciones primitivas o “pasos” ejecutados.

Apéndice K: Notación utilizada

- $\log n$ denota el logaritmo base 2 de un entero positivo n.
- a mod p denota la operación a módulo un número primo p.
- $\mathbb{R}$ denota el conjunto de los números reales.
- $\mathbb{Z}$ denota el conjunto de los números enteros.
- $\mathbb{Z}_p$ denota el grupo aditivo de los números enteros módulo un número primo p.
- $\mathbb{Z}_p^*$ denota el grupo multiplicativo de los números enteros módulo un número primo p.
- G denota un grupo multiplicativo finito.
- λ denota la pendiente de una recta.
- F denota un campo de por lo menos dos elementos.
- E denota una curva elíptica sobre un campo F.
- $E\ \mathbb{Z}_p$ denota una curva elíptica sobre un campo $\mathbb{Z}_p$.
- P_1 denota un punto sobre una curva elíptica $E\ \mathbb{Z}_p$.
- mcd a, b denota el máximo común divisor de dos números enteros no negativos a y b.
- max a, b denota el mayor de y b.
- x denota la función piso de x.
- $\mathcal{O}\ g\ x$ denota la notación $\mathcal{O}$ de Landau utilizada para referirse a las funciones que están acotadas superiormente por la función $g\ x$.

Referencias

Referencias

[1] MOLLIN, Richard. *Advanced number theory with applications:* Estados Unidos de América: Chapman & Hall/CRC. 2010. p. 301 – 309.

[2] LAWERENCE C., Washington. *Elliptic Curves: number theory and cryptography:* Estados Unidos de América: Chapman & Hall/CRC. 2008. p. 9 – 16.

[3] ROSEN, Kenneth. *Matemática Discreta y sus Alicaciones:* Colombia: McGraw – Hill/Interamericana de España. 2004. p. 120 – 126.

[4] MENEZES, Alfred J. *Handbook of applied cryptography:* Estados Unidos de América: CRC Press. 1997. p. 103 – 105.

[5] SCHNEIER, Bruce. *Applied cryptography:* Estados Unidos de América: John Wiley & Sons. 1996. p. 476 – 479.

[6] KATZ, Jonathan. *Introduction to modern cryptography:* Estados Unidos de América: Chapman & Hall/CRC. 2003. p. 282 – 287.

[7] WAGSTAFF, Samuel S. *Cryptanalisys of number theorectic ciphers:* Estados Unidos de América: Chapman & Hall/CRC. 2008. p. 171 – 182.

[8] JOUX, Antoine. *Algorithmic Cryptanalysis:* Estados Unidos de América: Chapman & Hall/CRC. 2009. p. 216 – 219.

[9] STAMP, Mark. *Applied cryptanalysis: breaking ciphers in the real world:* Estados Unidos de América: John Wiley & Sons. 2007. p. 275 – 283.

[10] FÚSTER SABATER, Amparo. *Técnicas criptográficas de protección de datos:* México: Alfaomega. 2001. p. 140 – 146.

[11] HERRERA CAMACHO, Abel. *Álgebra lineal: teoría y ejercicios:* México: Facultad de Ingeniería, Universidad Nacional Autónoma de México. 1987. p. 81 – 97.

[12] SOLAR GONZÁLEZ, Eduardo. *Apuntes de álgebra lineal:* México: Limusa. 2006. p. 483 – 530.

[13] GOBRAN, Alfonse. *Álgebra elemental:* México: Iberoamericana. 1990. p. 6, 7.

[14] *De Vita Caesarum, Divus Iulius (The Lives of the Caesars, The Deified Julius)* [en línea]. Universidad de Fordham. [Citado noviembre 28, 2011]. Disponible

de World Wide Web: http://www.fordham.edu/halsall/ancient/suetonius-julius.asp

[15] SHANNON, Claude. *Communication Theory of Secrecy Systems* [en línea]. Network Research Laboratory, Departamento de Ciencias de la Computación – Universidad de California, Los Ángeles. [Citado noviembre 29, 2011]. Disponible de World Wide Web: http://netlab.cs.ucla.edu/wiki/files/shannon1949.pdf

[16] WHITFIELD, Diffie. *New Directions in Criptography* [en línea]. IEEE Transactions on Information Theory, volúmen 22, número 6, noviembre de 1976 [citado julio 22, 2011]. Disponible de World Wide Web: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1055638&tag=1

[17] ELGAMAL, Taher. *A public key cryptosystem and a signature scheme based on discrete logarithms* [en línea]. IEEE Transactions on Information Theory, volúmen 31, número 4, julio de 1985 [citado abril 10, 2012]. Disponible de World Wide Web: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1057074

[18] J. L. Massey y J. K. Omura. *Method and apparatus for maintaining the privacy of digital messages conveyed by public transmition* [en línea]. U.S. Patent #4,567,600, 28 de enero de 1986. [Citado enero 31, 2012]. Disponible de World Wide Web: www.google.com/patents/US4567600.pdf

[19] *What is Diffie-Hellman?* [en línea]. RSA Laboratories. [Citado julio 22, 2011]. Disponible de World Wide Web: http://www.rsa.com/rsalabs/node.asp?id=2248

[20] RUÍZ DUARTE, Eduardo. *Introducción a las curvas elípticas, hiperelípticas y libcurve* [en línea]. Octubre de 2009 [citado julio 21, 2011]. Disponible de World Wide Web: http://math.co.ro/colfinal/coloquio-ce-ch.pdf