Introducción

Está disciplina matemática es el corazón de los procesos de validación y de verificación de las informaciones intercambiadas en una red, es lo que permite garantizar la confianza. En los siguientes apartados, vamos a describir los conceptos matemáticos subyacentes y su uso para garantizar y autentificar la información.

Históricamente, la criptografía se asocia con problemáticas que permiten realizar y analizar esquemas de encriptados (lo que autoriza el intercambio de informaciones secretas mediante canales de comunicación no protegidos). Sin embargo, a partir de los años 70, también se ha incluido en la criptografía la construcción de firmas electrónicas inviolables y el desarrollo de protocolos de tolerancia de errores. En este caso, la criptografía afecta a todo sistema que necesita enfrentarse a abusos o a ataques malintencionados. Nuestra definición de la criptografía hace que sea esencial y necesario el uso de determinadas herramientas matemáticas que se tratarán en este capítulo, como las funciones de sentido único, los generadores pseudoaleatorios y la prueba de divulgación sin conocimiento.

Es importante destacar que la criptografía se basa en una hipótesis muy importante: la existencia de las funciones de sentido único. Estas funciones capturan las dificultades de cálculo inherentes a la criptografía y las capitalizan en estas limitaciones de cálculo. Sin limitaciones de cálculo, este planteamiento de la criptografía no es operativo. Las dificultades generadas por estos problemas constituyen una oportunidad para la criptografía, pero una dificultad adicional para los algoritmos que permiten generar las funciones de sentido único.

# Esquemas de encriptado

El intercambio de información secreta a través de los canales públicos de comunicación representa el problema de criptografía más básico y tradicional. Esto consiste en que dos partes se comuniquen en secreto sin que una tercera parte utilice esta información. Habitualmente, un esquema de encriptado está formado por un par de algoritmos. El primero es el algoritmo de encriptado aplicado por el emisor del mensaje y el segundo es el algoritmo de desencriptado utilizado por el destinatario para leer el mensaje original. En esta configuración, por el canal de comunicación solo viaja el texto cifrado. Para garantizar el carácter secreto de este tipo de intercambio, las dos partes tienen que compartir una información que solo ellas poseen. Esta información adicional o «extra knowledge» puede tener la forma de un algoritmo de desencriptado o de un parámetro auxiliar utilizado por el algoritmo de desencriptado. Esta información adicional se llama clave de desencriptado. Podemos observar que el algoritmo de desencriptado puede ser conocido por todos, porque para descifrar el contenido hay que estar en posesión de la clave de desencriptado. También es importante destacar que la clave de desencriptado siempre debe mantenerse en secreto.

La evaluación del nivel de seguridad de este tipo de esquema no es tarea fácil, y en algunos casos resulta muy complicada o incluso imposible. Es importante definir el concepto de seguridad y su perímetro. Hay dos planteamientos posibles:

* El primero deriva de la teoría de la información, está relacionado con informaciones sobre el texto de origen contenidas en el texto cifrado. Si hay información sobre el texto de origen contenida en el texto encriptado, entonces el esquema de encriptados se considera como no protegido. También se ha demostrado que, para tener un nivel de seguridad alto, hace falta al menos una clave de encriptados tan larga como el texto de origen. Es una condición drástica que representa una limitación para este esquema, especialmente cuando se desea encriptar una gran cantidad de información.
* El segundo planteamiento es más moderno y se basa en el nivel de complejidad del cálculo (teoría de la complejidad). En este caso, no es importante saber si el texto cifrado contiene información sobre el texto de origen. La única pregunta que hay que hacerse está relacionada con la posibilidad de extraer información del texto. Dicho de otro modo, nos interesamos por el grado de complejidad para descifrar las informaciones encriptadas. En este contexto la longitud de la clave de encriptado no es un elemento «clave» en la seguridad del cifrado. Por ejemplo, podríamos utilizar un generador pseudoaleatorio de claves de encriptado tan largas como sea posible a partir de claves más cortas.

La teoría de la complejidad nos lleva a introducir algunos conceptos como el esquema de encriptado por clave pública. Antes de explicar este procedimiento, hay que destacar que a un algoritmo de encriptado se le puede incorporar un parámetro auxiliar que es la clave de encriptado. Esto significa que, para cifrar un mensaje, al algoritmo de encriptado también hay que darle la clave de encriptado. La mayoría de los esquemas de cifrado utilizan claves de encriptado y desencriptado iguales. Entonces surge el problema de la distribución de claves. Por eso se ha propuesto un esquema de encriptado nuevo basado en la teoría de la complejidad, donde la clave de encriptado (clave pública) puede ser conocida por todos y es distinta de la clave de desencriptado que se mantiene en secreto. Es imposible remontarse a la clave de desencriptado a partir de la clave de encriptado. Este esquema resuelve el problema de distribución de claves y permite hacer públicas las claves de encriptado (identificadores).

Este tipo de esquema de encriptado se utiliza en los sistemas distribuidos como blockchain, para controlar los accesos y la validez de la información que circula en la red.

# Generadores pseudoaleatorios

Los generadores pseudoaleatorios desempeñan un papel principal en la construcción de los esquemas de encriptado. En concreto, permiten generar claves de encriptado privadas. Sin embargo, hay que decir que el término «generador pseudoaleatorio» también se utiliza en otros contextos, como los procedimientos probabilísticos. Por lo tanto, es primordial dar una definición clara y precisa de su importancia en criptografía.

Los generadores pseudoaleatorios son algoritmos deterministas que permiten extender una secuencia de números aleatorios “semilla”, para obtener una secuencia de bits más larga y que parece aleatoria aunque no lo es. La característica de pseudoaleatorio y la teoría de la complejidad están relacionadas en esencia, porque estos generadores se construyen sobre la hipótesis de insolubilidad (*intractability assumption*). La hipótesis de la existencia del generador pseudoaleatorio está vinculada a la existencia de funciones de sentido único, porque está construido a partir de funciones de sentido único particulares.

# Fundamentos de la teoría de las probabilidades

En este apartado vamos a definir las funciones de sentido único haciendo hincapié en sus descripciones matemáticas. También propondremos ejemplos de estas funciones y de sus aplicaciones.

Antes de presentar estas funciones de sentido único, vamos a recordar algunos conceptos sobre los cálculos de probabilidades. Estos fundamentos son la base de las funciones de sentido único y de la teoría de la complejidad.

Las probabilidades desempeñan un papel determinante en criptografía. Son especialmente importantes para tratar la información o la falta de ella (el secreto). En esta parte, abordamos algunos conceptos y desigualdades propias de la criptografía.

De ahora en adelante, solo nos referiremos a las distribuciones de probabilidades discretas. En nuestro caso, el espacio de probabilidad consiste en una secuencia de caracteres distribuidos de manera uniforme y de una longitud determinada . Es una muestra de una longitud total de caracteres de  bits, y cada carácter tiene una probabilidad asignada de .

Las variables aleatorias son funciones que asignan un valor del espacio de muestra al espacio real. La variable aleatoria representa el conjunto de los resultados definidos en el conjunto de las eventualidades.

*Ejemplo*

Dada una variable aleatoria  y una expresión booleana  dependiente de la variable , entonces la probablidad  significa que  se verifica cuando se elige  con la probabilidad .

En otras palabras:



donde  es la función indicadora, lo que implica que  si tiene lugar el acontecimiento , y 0 en el caso contrario. Y tenemos que  para todas las variables aleatorias .

Este resultado se puede generalizar a dos variables aleatorias independientes y distribuidas de manera idéntica,  e . Entonces tenemos , la probabilidad de que  tenga lugar para el par  elegido con una probabilidad :



Por ejemplo,  solo si  e  son triviales (toda la masa de probabilidad se asigna a un único carácter).

En criptografía, la variables aleatoria uniformemente distribuida es una variable aleatoria muy útil , que designa una distribución de una secuencia de caracteres en una longitud . La probabilidad es  si , y 0 en caso contrario.

De ahora en adelante van a ser muy útiles tres desigualdades fundamentales en teoría de probabilidades, porque permiten establecer límites superiores para las variables aleatorias. Se trata de la desigualdad de Markov, la desigualdad de Tchebychev y la desigualdad de Hoeffding (Stein, 2005).

**Desigualdad de Markov**

Esta desigualdad demuestra que para una variable aleatoria con un límite inferior o superior, existe una relación entre la desviación de este valor de la esperanza de la variable aleatoria y la probabilidad de que esta variable aleatoria sea superior o inferior a este valor.

Dada una variable aleatoria positiva  y un número real . Entonces:



De manera equivalente: 

Se puede demostrar como se describe a continuación:







La desigualdad de Markov es muy útil cuando conocemos pocos parámetros de la distribución de la variable aleatoria. Entonces es suficiente con conocer su esperanza y, al menos, uno de los extremos o límites del intervalo de definición de sus valores.

**Desigualdad de Tchebychev**

Dada una variable aleatoria  y . Entonces:



Si  es la varianza de , observamos que .

Para demostrar esta desigualdad, definimos la variable  y aplicamos la desigualdad de Markov. El resultado es:





Esta desigualdad es muy interesante y muy útil en el análisis de los errores de probabilidad de las aproximaciones. Entonces solo hay que suponer que las muestras se han extraído de manera que son independientes por pares.

**Corolario**: dadas las variables aleatorias independientes por pares  con la misma esperanza  y la misma varianza . Para todo :

Diagrama, Esquemático

Descripción generada automáticamente

Se dice que las  variables aleatorias son independientes por pares si para todo  y para cualesquiera  y ,  es igual a .

Utilizando una muestra independiente por pares, los errores de probabilidad en la aproximación decrecen de manera lineal con la cantidad de puntos de la muestra. En el caso en que los puntos de la muestra son completamente independientes, asistimos a un decrecimiento exponencial del error con la cantidad de puntos en la muestra.

Si  y  son variables aleatorias discretas {1;0} e independientes, con una probabilidad  para cada *i*. Para todo , tenemos:

Diagrama, Esquemático

Descripción generada automáticamente

Esta expresión representa el límite de Chernoff, nos da el valor máximo de la probabilidad de que una variable aleatoria supere un valor fijo. Permite aumentar la cola de una distribución de una ley de probabilidad. Es análoga a la desigualdad de Markov, pero con un límite exponencial.

En la práctica, utilizamos una probabilidad . En este caso, la desviación respecto a la esperanza es una probabilidad que decrece de manera exponencial con . Con frecuencia, esta aproximación se representa por -aproximación y se puede alcanzar utilizando una muestra de  puntos.  nos da información sobre el error de probabilidad y  sobre la precisión del estimador (con un límite superior).

**Desigualdad de Hoeffding**

Dadas las variables aleatorias independientes  con la misma distribución en el intervalo  y la esperanza de cada variable aleatoria . Para todo :

Diagrama

Descripción generada automáticamente con confianza media

A menudo, esta desigualdad se utiliza para estimar el valor medio de una función definida en una amplia gama de valores y con errores de probabilidad despreciables. También se aplica para tener una muestra eficaz con límites en el interior de los valores posibles (Stein, 2005).

# Modelos de cálculo (complejidad)

El planteamiento de la criptografía se basa en gran medida en la teoría de la complejidad. Por eso creemos que es necesario hacer algunos recordatorios sobre el cálculo de la complejidad y sus distintas clases.

La teoría de complejidad es un campo de las matemáticas aplicado a la informática, se trata de la informática teórica. Esta disciplina se interesa por la complejidad de un algoritmo y por los recursos en términos de tiempo y memoria necesarios para su ejecución y realización. Por lo tanto, se trata de analizar las propiedades intrínsecas de un problema y estudiar su complejidad. Estos problemas se distribuyen en clases de complejidad.

En función de la respuesta esperada, hay dos tipos de problemas: los problemas de decisión con una respuesta esperada binaria (sí o no). También se plantean problemas de existencia o de búsqueda de soluciones con una respuesta que consiste en proporcionar un elemento como respuesta a un requerimiento. En la teoría de la complejidad, nos interesamos principalmente por los problemas de decisión.

La cuantificación de la complejidad depende en gran medida de la representación de los datos en la entrada de la instancia del problema. Tomando el ejemplo del número entero 10 como instancia de un problema. Este número entero se puede representar en unario (||||||||||) o en binario (1010). El tamaño es distinto en las dos representaciones, 10 (cifras decimales) para una y 4 para la otra.

Para estos problemas se utilizan algoritmos que ejecutan etapas elementales de cálculo, la cantidad depende del tamaño a la entrada de la instancia. La complejidad en términos de tiempo, es el tiempo de ejecución de un algoritmo proporcional a la cantidad de etapas necesarias en la resolución de un problema. La teoría de la complejidad se centra en el cálculo de la dificultad intrínseca de un problema algorítmico, porque para resolver el mismo problema pueden coexistir muchos algoritmos.

De todo esto surge una idea muy importante: la longitud de un número. La complejidad de un número depende en gran medida de esta longitud. Es la dimensión de los números en una base determinada, sobre la que opera un algoritmo.

Se llama longitud  de un número  en base , la cantidad  tal que:



Expresado de otra manera, .

De la ecuación anterior se deriva: .

Lo que implica:  y .

Podemos hacer una estimación de la longitud independientemente de la base. Hay una relación logarítmica entre las distintas bases:



Si representamos el logaritmo en la base natural como  sin índice, entonces tenemos: .

Utilizando esta relación obtenemos: , sabiendo que la base  donde se expresan los números se fija al inicio y se convierte en una constante.

También es necesario definir un vocabulario determinado. Se dice que un problema es resoluble si podemos expresar la respuesta mediante un algoritmo y que es calculable si podemos calcular la solución mediante un algoritmo. La teoría de la complejidad se define en este marco, porque solo trata los problemas resolubles y calculables, y busca estimar las necesidades de tiempo y memoria de un algoritmo.

La teoría de la complejidad establece una jerarquización de las dificultades entre los problemas algorítmicos. Los distintos niveles se denominan clases de complejidad. Esta clasificación se ha establecido en función de los tipos de cálculos: determinista (el siguiente estado del cálculo se determina mediante el estado actual) y no determinista. Para la determinación de la complejidad se utilizan modelos de cálculo. Los cuantificadores clásicos de la complejidad son el tiempo y el espacio, pero hay otros cuantificadores de recursos (en el campo de la comunicación, por ejemplo).

Uno de los modelos más conocidos es el de las máquinas abstractas, como la propuesta por Alan Turing en 1936, pero también podemos citar la máquina RAM (*RandomAccess Machine*, Máquina de acceso aleatorio). En los dos modelos, en cada una de las etapas elementales, para un estado dado de la memoria de la máquina, se elige una acción elemental entre el conjunto de las posibles acciones. En el caso de una maquina determinista, el estado actual de la máquina determina la siguiente acción. Por otro lado, si hay varias opciones posibles, se dice que la máquina es no determinista. La potencia de una maquina se mide por la cantidad de problemas que puede resolver en un tiempo fijo. Sin embargo, hay otros modelos de cálculo.

## **1. Clases de complejidad**

Una clase de complejidad representa un conjunto de problemas cuya resolución demanda la misma cantidad de un recurso particular en términos de tiempo y espacio. Podemos citar el ejemplo de las clases *P*, *NP* y *co - NP*. Hay dos categorías de clases:

**Clases limitadas por el tiempo**

* *P*: es la clase de los problemas que resuelven en tiempo polinómico por una máquina determinista. Es la clase de los algoritmos eficaces (Garey, 1979).
* *NP*: es la clase de los problemas que se resuelven en tiempo polinómico por una máquina no determinista.
* *EXPTIME*: es la clase de los problemas resueltos en tiempo exponencial por una máquina determinista.
* *NEXPTIME*: es la clase de los problemas resueltos en tiempo exponencial por una máquina no determinista.

**Clases limitadas por el espacio**

* *L*: grupo de problemas algorítmicos resueltos en espacio logarítmico y en máquinas deterministas.
* *NL*: grupo de problemas algorítmicos resueltos en espacio logarítmico y en máquinas no deterministas.
* *PSPACE*: grupo de problemas algorítmicos resueltos en espacio polinómico y en máquinas deterministas. Según el teorema de Savitch, esta clase es equivalente a la clase *NPSPACE* (Garey, 1979).
* *EXPSPACE*: grupo de problemas algorítmicos resueltos en espacio exponencial y en máquinas deterministas. Según el teorema de Savitch, esta clase es equivalente a la clase *NEXPSPACE*.

**Inclusión de las clases**



**Reducción polinómica**

Es un concepto muy importante en la teoría de la complejidad, Especialmente para los problemas *N = NP*. Dado un lenguaje formal *L1*, (el conjunto de las estancias cuya respuesta es *SÍ*) para los problemas de decisión, *L1* se puede reducir a *L2* con  en un tiempo polinómico, si existe una función polinómica calculable  tal que para todo , , si y solo si . La función  se llama función de reducción, y el algoritmo polinómico que calcula  se llama algoritmo de reducción.

## **2. Algoritmos y complejidad**

Vamos a determinar la complejidad de algunos algoritmos, como los algoritmos euclidianos. El algoritmo euclidiano en cuestión es el máximo común divisor (MCD) que permite determinar el máximo común divisor entre dos números enteros  y . Una primera estimación aproximada de la cantidad de etapas necesarias para este algoritmo daría . Basándonos en esta estimación, deducimos que la cantidad de divisiones necesarias para la ejecución de este algoritmo evoluciona en tiempo exponencial, lo que representa un punto muerto porque se plantea la cuestión de la viabilidad. La complejidad sería del orden de:



donde  es una estimación del coste de cada división.

Por lo tanto, es importante mejorar la cantidad de divisiones . Para ello, tomemos el caso donde  y  son dos números consecutivos de una serie de Fibonacci. En este proceso es necesario recordar la siguiente proposición :

**Proposición**: sea  el número n-ésimo de una serie de Fibonacci. La cantidad de divisiones necesarias para calcular  utilizando un algoritmo euclidiano es *n*-1.

Demostración: tenemos

Texto, Carta

Descripción generada automáticamente

Como , tenemos , que es exactamente *n*-1 divisiones calculadas por el algoritmo para encontrar el máximo común divisor.

El ejemplo tiene como datos de entrada (input) el par de números  y el algoritmo sigue el diagrama: .

Sin embargo, para cualquier otra elección de pares de números  con , considerando que el algoritmo euclidiano se termina después de  etapas, estará delimitada por un par de una serie de Fibonacci . Esto se debe al teorema de Lamé (Halton, 1996).

**Teorema**: dados dos números enteros positivos  y  con  y dada . Si el algoritmo euclidiano para calcular  termina después de  etapas, entonces  y .

Para la demostración de este teorema consulte (Baldoni, Ciliberto, Cattaneo, 2009; Halton, 1996).

Este teorema permite determinar un límite superior para . Eso nos permite formular la siguiente proposición: dados dos números enteros positivos  y  con  entonces:



porque  para todo  si .

La ecuación anterior permite tener una estimación de la cantidad de divisiones necesarias en la ejecución del algoritmo, y es mucho mejor que nuestra estimación de partida. Ahora tenemos:



porque:



con  la estimación de la complejidad relacionada con el número de divisiones y  la estimación de la complejidad de cada división.

Por lo tanto, la complejidad de un algoritmo euclidiano es polinómica.

Ahora, vamos a analizar un ejemplo para ilustrar estas afirmaciones. Aplicaremos el algoritmo euclidiano para el par (34567,457). El número  tiene tres cifras, entonces . Dicho de otro modo, el algoritmo finalizará al cabo de divisiones como máximo.

34567 = 457.75 + 292

457 = 292.1 + 165

292 = 165.1 + 127

165 = 127.1 + 38

127 = 38.3 + 13

38 = 13.2 + 12

13 = 12.1 + 1

12 = 1.12 + 0

Así, se necesitan ocho operaciones (<15) para verificar que MCD (34567,457) = 1.

**Corolario**: la conclusión de estos resultados es que la complejidad de un algoritmo para encontrar una solución de una ecuación bajo la forma de  es , donde  es la mayor longitud entre los números enteros .

Cuando trabajamos en base 2, la conversión a base decimal es:



De ello, deducimos que para convertir un número  de una base  a una base  existe un algoritmo de complejidad  con dos números enteros superiores a 1.

A partir de los resultados anteriores es posible determinar la complejidad de las operaciones en los polinomios. Dados dos polinomios  y  con coeficientes  tales que . Suponemos que los coeficientes tienen una longitud máxima de . Entonces:

* La suma o la diferencia de  y  tiene una complejidad de .
* El producto de  y  tiene una complejidad de .

Puede encontrar la prueba de estas proposiciones en (Baldoni, Ciliberto, Cattaneo, 2009).

# Tiempo polinómico

El tiempo polinómico se pude considerar como el tiempo mínimo de ejecución de un algoritmo en función de los datos de entrada. Un algoritmo se resuelve en tiempo polinómico si , para todas las constantes *c* y *n* independientes, siendo *n* un número entero que representa el tamaño de los datos de entrada, se ejecuta en menos de  operaciones elementales (y  es una constante independiente de las otras dos).

El tiempo polinómico también se puede definir utilizando la longitud de los números en una base dada, utilizados por un algoritmo. Dado un algoritmo A que realiza cálculos sobre un número entero. Se dice que este algoritmo es polinómico o en tiempo polinómico si existe un número entero positivo  para el que la cantidad de operaciones elementales en bits necesarias para la ejecución del algoritmo es inferior o igual a .

En caso de que la cantidad de operaciones necesarias para la ejecución del algoritmo sea inferior a  para una constante *c* superior a 0, entonces se dice que el algoritmo es exponencial. El ejemplo típico de un algoritmo en tiempo exponencial es la criba de Eratóstenes, que permite encontrar los números primos inferiores a un determinado número natural *N*. Es un algoritmo que procede por eliminación. Elimina todos los múltiples de un número entero de una tabla de números enteros hasta *N*. Al final, solo quedan los números primos que no son múltiples de ningún número entero.

La diferencia entre tiempo polinómico y tiempo exponencial es la siguiente: supongamos que trabajamos con números grandes, de tal manera que sea imposible trabajar con ellos manualmente o en una hoja de papel, con un lápiz. Entonces, si un algoritmo es polinómico, ejecutar el cálculo con una máquina adaptada costará un tiempo “razonable”. Y si el algoritmo es de tiempo exponencial, en general no podemos obtener una respuesta en un tiempo “razonable”, ni siquiera con una máquina de cálculo muy potente. Es decir, los algoritmos polinómicos son factibles o realizables y los algoritmos exponenciales no lo son.

Tomemos un ejemplo para ilustrar estas afirmaciones. Dado el tiempo *t* necesario para hacer un cálculo con un bit de operación o una sola operación. Si un algoritmo es polinómico de orden 2, y si queremos utilizarlo para cálculos de números muy grandes del orden de 10100, entonces el tiempo necesario para realizar este cálculo será 10000*· C · t*, donde *C* es una constante adecuada. Hay que señalar que *t* depende de la potencia de la máquina utilizada para el cálculo. Para una máquina de cálculo estándar, *t* es 10-9 segundos, lo que da 10000*· C · t* = 10-5*C* segundos. También observaremos que cuanto mayor es el orden del algoritmo, más importante es su tiempo de ejecución. Un algoritmo de orden 10 aplicado al mismo ejemplo daría un tiempo de cálculo de 1011 *· C* segundos.

Con un algoritmo exponencial, el tiempo que costaría realizar el mismo cálculo sería 10100*· t* segundos. Incluso teniendo el ordenador más potente del mercado, siendo *t* 10-20 segundos, el tiempo de cálculo seguiría siendo astronómico del orden de 1080 segundos. Haciendo una comparación, en el universo existen unos 1079 protones (partículas que forman el núcleo de un átomo). Este tipo de algoritmo no se utiliza nunca para números tan grandes. Sin embargo, los algoritmos exponenciales son más eficaces para números pequeños. El mejor ejemplo es el algoritmo de criba de Eratóstenes, muy utilizado para números enteros naturales pequeños. La figura 3.1 representa una ilustración de la diferencia de amplitud entre el tiempo polinómico y el tiempo exponencial.

Gráfico

Descripción generada automáticamente con confianza media

*Figura 3.1 - Ilustración de la evolución en tiempo polinómico y en tiempo exponencial*

Ahora que se ha dado una primera definición del tiempo polinómico, es necesario introducir otro concepto: los algoritmos probabilísticos permiten generar secretos.

## **1. Algoritmos probabilísticos**

Un algoritmo que utiliza una fuente aleatoria se denomina algoritmo probabilístico o incluso aleatorizado. La ejecución del algoritmo hace intervenir extracciones aleatorias de datos de los que depende la naturaleza de la acción que se ejecuta. La naturaleza de los datos extraídos de manera aleatoria depende de la ley de distribución previamente elegida como hipótesis. Son algoritmos sencillos de analizar y rápidos respecto a otros algoritmos deterministas, pero también de datos aleatorios. Por lo tanto, con los mismos datos deterministas, podemos tener dos resultados completamente distintos como consecuencia de los valores aleatorios.

Para ilustrar estas afirmaciones, tomemos el ejemplo de un algoritmo probabilístico aplicado a un grafo (teoría de grafos). El algoritmo permite saber si dos nudos de un grafo están conectados.

El algoritmo toma en la entrada un grafo *G = (V,E)* (figura 3.2) y dos vértices *s* y *t*. Utilizamos el funcionamiento aleatorio para extraer longitudes del orden de . Empezamos por *s*, en cada etapa probamos si se encuentra (alcanza) el vértice *t*. Si no se encuentra nunca el vértice *t*, entonces el algoritmo acepta la hipótesis nula (el nudo no está conectado), si no la rechaza. El funcionamiento aleatorio permite seleccionar una arista incidente al vértice que recorre y atraviesa otro nudo. Si no está conectada a *t* en el grafo *G*, entonces la probabilidad de que el algoritmo acepte la hipótesis es despreciable. Este tipo de algoritmo también se utiliza para calcular intervalos de confianza.

Esquemático

Descripción generada automáticamente

*Figura 3.2 - Ejemplo de un algoritmo probabilístico aplicado a un grafo. El algoritmo permite saber si dos nudos del grafo están conectados basándose en un funcionamiento aleatorio.*

Hay dos puntos de vista sobre los algoritmos randomizados. El primero es el hecho de que el algoritmo permite hacer movimientos aleatorios en cada etapa de su ejecución. Es el caso de máquina de Turing, donde la función de transición (*<estado>*,*<símbolo>*) tiene dos posibilidades en cuanto a la elección de los tripletes (*<estado>*,*<símbolo>*,*<dirección>*). La elección se hace de manera aleatoria, es decir, en cada etapa nueva tenemos una probabilidad (1/2 en este caso) de elegir una u otra dirección. El output (salida) de una máquina probabilística *M* con entrada *x* es una variable aleatoria . El espacio de probabilidad representa todas los outputs posibles para un proceso aleatorio de distribución uniforme. Con la hipótesis de algoritmo en tiempo polinómico, podemos considerar, sin perder el alcance general, que la cantidad de extracciones aleatorias en el input  es independiente del output . La notación  define el output de *M* en el input *x* cuando *r* es la salida de las extracciones aleatorias internas. Entonces, la probabilidad  es simplemente la fracción de  para la que .

Entonces tenemos:



La otra manera de ver los algoritmos randomizados es considerar las extracciones aleatorias internas en la salida como un input. Es decir, consideramos una máquina determinista con dos inputs , donde el segundo input es la secuencia de extracciones al azar, internas. El input *r* se elige de manera uniforme en el intervalo . Esquemáticamente, las extracciones aleatorias no son internas, sino que un generador externo se las suministra a la máquina.

En esta fase es importante observar que las máquinas no deterministas son en cierto modo ficticias e irrealizables, solo se utilizan para problemas de investigación. Mientras que las máquinas basadas en modelos probabilísticos son realistas y utilizables para cálculos concretos.

## **2. PP y BPP**

En el ámbito de la teoría de la complejidad, *PP* (*Probabilistic Polynomial time*) es un objeto que agrupa al conjunto de los problemas con una decisión calculada por una máquina de Turing probabilista en tiempo polinómico, y con una probabilidad de error inferior a 1/2. Es decir, un idioma *L* está en la clase PP si existe una máquina de Turing *M* tal que:

* *M* termina en tiempo polinómico en todas las instancias.
* *M* acepta todas las palabras de *L* con una probabilidad superior a ½.
* *M* rechaza todas las palabras que no están en *L* con una probabilidad superior a ½.

La clase *BPP* (*Bounded-error Probabilistic Polynomial time*) es muy similar a la clase *PP*, pero se considera una clase diferente. La clase *BPP* trata la categoría de problemas que se pueden decidir utilizando una máquina probabilista en tiempo polinómico con una probabilidad de error inferior a una constante inferior a ½. Esta clase de problemas satisface las siguientes condiciones:

* La máquina rechaza todas las palabras que no están presentes en el idioma con una probabilidad de 2/3.
* La máquina acepta todas las palabras presentes en el idioma con una probabilidad de 2/3.

En este caso, el índice de error es de 1/3.

## **3. Funciones despreciables**

Una función  de aplicación de  se considera despreciable si, para todos polinomio , existe *N* tal que para todo :



Entre los ejemplos más conocidos de las funciones despreciables, tenemos . La multiplicación de una función despreciable por cualquier polinomio no induce ninguna modificación en su trayectoria. Eso tiene una implicación directa en la interpretación de los eventos. Porque si la probabilidad de que un evento se produzca es despreciable, entonces será verdad si repetimos la experiencia un gran número de veces.

## **4. Hipótesis de insolubilidad (intractability assumptions)**

Se considera «intractable» o insoluble todo problema o tarea que no puede resolverse utilizando una máquina probabilística en tiempo polinómico. El enfoque criptográfico de este tipo de problema solo es interesante si *NP* (*Nondeterministic Polynomial time*) no está contenido en *BPP*, lo que implica .

Cuando *<insolubilidad>* implica *<consecuencia útil>*, también se verifica la siguiente hipótesis: *<consecuencia útil>* implica *<insolubilidad>*.

En teoría de la computabilidad, hay una variante de las máquinas de Turing: la máquina “Oracle”. Una máquina capaz de resolver problemas de decisión en una sola operación, sea cual sea la complejidad de este problema.

## **5. Funciones de sentido único**

Esta es la manera más sencilla de definir una función de sentido único: una función fácil de calcular pero muy complicada de invertir. La primera condición es muy clara: al decir función "fácil de calcular" está implícito que existe un algoritmo polinómico para cualquier entrada *x* que da como salida *f(x)*. La segunda condición es más elaborada. En la expresión “difícil de invertir” está implícito que todo algoritmo polinómico que intenta encontrar su inverso aplicado en un input *y* bajo la función  tendrá una probabilidad de éxito muy baja, la probabilidad se considera sobre todas las elecciones de *y*.

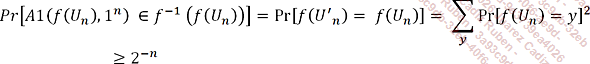
Por definición, una función  se dice que es de sentido único si se cumple dos condiciones:

* Fácil de calcular: existe un algoritmo polinómico determinista *A* como .
* Difícil de invertir: para todo algoritmo probabilista polinómico , todo polinomio  y todo  suficientemente extenso, , donde  es una variable aleatoria distribuida de manera uniforme en , porque la probabilidad se calcula sobre todos los valores posibles de . Observemos que no necesitamos  para construir una imagen previa de . El input auxiliar  es una convención sobre la longitud del output producto del algoritmo .

La dificultad de inversión se puede interpretar como un límite superior a la probabilidad de éxito de un algoritmo de inversión eficaz (las desigualdades). Esta probabilidad se cuantifica considerando la elección aleatoria del algoritmo de inversión y la distribución del input de este algoritmo (). Esta distribución se obtiene aplicando  al  seleccionado de manera uniforme en .

Para explicar estas proposiciones, vamos a considerar los dos algoritmos simples (triviales) que se mencionan a continuación:

**El algoritmo "random guess"** (adivinanza aleatoria) *A*1: aplicado a un input , el algoritmo *A*1 seleccionado de manera uniforme y da como resultado un carácter de longitud . La probabilidad de éxito del algoritmo *A*1 es igual a la probabilidad de colisión en la variable aleatoria . Dada una variable aleatoria  distribuida de manera uniforme en  e independiente de , entonces tenemos:



Entonces:

* para toda función , la probabilidad de éxito de un algoritmo trivial es estrictamente positiva (por definición). Es imposible suponer que todo algoritmo eficaz va a fracasar al invertir la función ;
* la probabilidad de éxito de un algoritmo trivial al invertir  es despreciable. Sin embargo, eso no implica que  sea una función de sentido único;
* si  es una función de sentido único, entonces la probabilidad de colisión es despreciable;

**El algoritmo "fixed output"** (salida fija) *A*2: este otro algoritmo trivial calcula una función constante para todo input de la misma longitud ( (Goldschmidt, 1964).

Para toda función :



Por lo tanto:

* una vez más, la probabilidad de éxito es estrictamente positiva;
* la probabilidad de invertir la función  mediante un algoritmo trivial *A*2 es despreciable.

Las funciones de sentido único anteriormente descritas son funciones de sentido único en el sentido "estricto" del término. Sin embargo, existen otras funciones de sentido único con más probabilidades de inversión (porque hay algoritmos más eficaces para invertir las funciones de sentido único). Son las funciones de sentido único en el sentido "débil".

Dada una función . Esta función se denomina de sentido único débil si:

* es fácil de calcular;
* es "ligeramente" difícil de invertir: para todo algoritmo probabilístico polinómico , todo polinomio  y todo  suficientemente grande, .

Hay otras funciones de sentido único que tienen una grieta secreta y permiten que cualquiera que la conozca las invierta con facilidad. Son funciones de sentido único con grieta secreta. Estas funciones se utilizan en la criptografía RSA, donde la clave privada representa la grieta. Otras funciones solo se definen para una cierta longitud de caracteres.

## **6. Candidatos para las funciones de sentido único**

Las funciones que mencionaremos a continuación son candidatos firmes para ser funciones de sentido único, porque son objeto de intensas investigaciones. En efecto, hasta ahora, siempre hemos fracasado en la producción de algoritmos eficaces para invertirlas. Pero eso no significa que estas funciones sean realmente de sentido único.

**Factorización completa en número primos**: en aritmética, la descomposición en producto de factores primos consiste en escribir un número natural no nulo en forma de producto de números primos. Vamos a tomar el ejemplo del número 25, que se puede representar como 5 · 5. Esta factorización se caracteriza por ser única, según el teorema fundamental de la aritmética. Además, facilita la manipulación de estos números en problemas complejos.

En la teoría de la complejidad, el mejor algoritmo de factorización conocido hasta el presente está en tiempo subexponencial. Por lo tanto, es bastante razonable afirmar que la función  que particiona un input en dos partes (en representación binaria) es de sentido único.



donde  y  designa al número entero (secuencia de caracteres) resultante del producto de las dos partes . Considerando la insolubilidad de la factorización (por ejemplo, es prácticamente imposible remontarse a los dos factores primos de la misma longitud en bits conociendo su producto) y el teorema de la densidad de los números primos (al menos  de los números enteros inferiores a *N* son primos), se deduce que  es como mínimo un sentido único débil.

**Problema de la suma de subconjuntos**: en algorítmica se trata de un problema de decisión. Se puede resumir de la siguiente manera: dado un subconjunto *E* de  enteros, existe un subconjunto de *E*, tal que la suma de los elementos es nula (Cormen, 2002). La respuesta puede ser sí o no:

Dada la función  definida de la siguiente manera:



donde  e . Vemos claramente que  es de tiempo polinómico. El hecho de que el problema sea complejo en *N - P* completo no implica de manera directa que la función es de sentido único. En efecto, esta función se considera de sentido único por la ausencia de algoritmos eficaces que puedan generar instancias de densidad muy elevada (instancias de números muy grandes (1 millón)).

**Decoding random linear codes (decodificación de códigos aleatorios lineales)**: en el campo de la corrección de errores de códigos, los algoritmos de decodificación de códigos aleatorios lineales representan un sector donde se llevan a cabo intensas investigaciones. En particular, los algoritmos aleatorios lineales, con un índice de información constante, capaces de corregir una constante fracción de errores. Un código  es un código lineal con una matriz binaria   en la que el vector suma (*mod 2*) cualquier subconjunto de líneas dentro de un vector con al menos *d* entradas de 1 (one-entries). Un mensaje de -bits se puede codificar multiplicándolo por la matriz  y el resultado será un vector de longitud -bits único. El límite de Gilbert-Varshamov para los códigos lineales garantiza la existencia de este tipo de código si  donde  si , y  en caso contrario.  es la función de modificación de la entropía binaria.

**Debate**: existe una función débil de sentido único si y solo si existe una función fuerte de sentido único (Silverman, 1985).

# Secretos … y verdades

¿Cómo transmitir información de manera que solo puedan acceder a ella las personas autorizadas? ¿Cómo asegurarse de que esta información llegará a su destinatario sin ser alterada o modificada? O incluso, ¿cómo podemos estar seguros del origen y veracidad de esta información? La respuesta a estas preguntas aportada por la criptografía permite la instauración de la confianza entre las distintas partes, esta misma confianza que es la esencia de los protocolos Blockchain.

A continuación, primero examinaremos los primeros intentos de cifrado y su evolución a lo largo de los siglos. Luego, nos interesaremos por la criptografía moderna y especialmente por las investigaciones en torno a la criptografía de clave pública que ese utiliza en los distintos protocolos Blockchain (pero no solo en ellos).

Para consultar un análisis en profundidad de la criptografía, consulte Shamir, A. «A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem.», 1982 (ver bibliografía).

## **1. Primeros cifrados (el cifrado clásico)**

La humanidad siempre ha necesitado métodos de comunicación de información sensible o relevante de manera secreta. ¿Cómo transmitir mensajes que solo pueden leer los destinatarios? Actualmente, este problema milenario es de una importancia capital, con el desarrollo de los sistemas electrónicos de comunicación que facilitan los intercambios y las iteraciones entre distintas entidades. Sin embargo, estos sistemas son vulnerables.

El ejemplo más antiguo de mensajes secretos se remonta a la Antigua Grecia y aparece en las *Historias* de Heródoto (libro *VII*), en sus descripciones de las guerras greco-persas.

**Primer mensaje secreto de la historia**

Heródoto es un narrador extraordinario, dotado de un talento increíble cuando se trata de describir y contar todo lo que ve, especialmente sus numerosos viajes a Asia Menor (Anatolia), Grecia, África, Sicilia, etc.

En su libro *VII*, cuenta cómo se transmite información o mensajes de manera secreta, especialmente en tiempos de guerra. Los procedimientos utilizados revelan más estenografía que criptografía, porque se disimulaba incluso la existencia de los mensajes. Así, Heródoto describe el procedimiento utilizado por Histieo, antiguo tirano, tío y suegro de Aristágoras, nuevo tirano de Mileto. Cuando se encontraba en Persia ejerciendo como consejero, quiso informar a su yerno de que había llegado el momento de sublevarse contra Persia. ¿Cómo avisarle si todas las carreteras estaban vigiladas? Eligió a un esclavo de leal, le afeitó la cabeza y escribió un mensaje en su cuero cabelludo. Cuando el pelo volvió a salir, hizo que el esclavo partiera hacia Mileto con la única instrucción de que le afeitaran la cabeza. El mensaje que llevaba el esclavo invitó a Aristágoras a la revuelta.

Cuando Demarato estaba exiliado en la corte de persa, utilizó tablillas para avisar al rey de Esparta de un ataque inminente. Tomó unas tablillas, retiró la cera y grabó el mensaje directamente sobre la madera. Luego las recubrió de cera. Mediante esta estratagema las tablillas parecían estar en blanco y no llamaron la atención. Cuando llegaron a Esparta, se las entregaron a la reina Gorgo que tuvo la idea de rascar la cera y leer el mensaje de Demarato. Así se convirtió en el primer desencriptador de la historia.

Gorgo salvó al mundo griego de los Persas utilizando su inteligencia, pero causó la muerte de su marido Leónidas en la batalla de las Termópilas.

Vamos a terminar este preámbulo histórico con una última anécdota. En el siglo XVI, María Estuardo de Escocia estaba prisionera en Inglaterra por orden de su prima Isabel I en el castillo de Chartley Hall. Desde allí participó en una conspiración, planeada por Anthony Babington, para matar a la reina Isabel. Se estableció una correspondencia entre María y Anthony especialmente destinada a utilizar ayuda extranjera proporcionada por los españoles. Y por supuesto, por motivos de seguridad, todos los intercambios estaban cifrados siguiendo un código creado por Morgan, fiel aliado de María Estuardo. Pero el primer secretario de la reina Isabel, ayudado por un agente doble, Gilbert Gifford, vigilaba toda la correspondencia del castillo Chartley Hall, y así el equipo de espionaje y criptoanálisis consiguió descifrar las cartas de María Estuardo derrotando al código, que de hecho era bastante sencillo. Se trataba de un código de sustitución: cada letra se sustituía por un símbolo y algunas palabras habituales también se sustituían por un símbolo. María Estuardo fue decapitada poco después.

La historia de María Estuardo nos enseña un principio fundamental de la criptografía: si no tenemos un código de cifrado fiable, como en este caso, es mejor no cifrar los mensajes.

Estos ejemplos históricos demuestran la importancia de intercambiar información de una manera segura, a espaldas de los enemigos y adversarios. Al contrario que la estenografía, la criptografía no busca ocultar los mensajes, sino el significado. El mensaje se puede leer, pero se supone que nadie es capaz de comprender su significado.

Estos primeros sistemas de cifrado basados en el cifrado del alfabeto donde cada letra se desplaza un número  en función de su posición se llaman cifrado de César o código de César. Es decir, hay 25 códigos de César posibles en función de la cantidad de posiciones desplazadas , que representan la clave de codificación (26 letras del alfabeto). En la actualidad, este código es muy fácil de desencriptar, pero en la época de César donde los instrumentos de cálculo no existían, este tipo de sistema se empleaba con mucha frecuencia.

Este código se perfeccionó añadiendo un cifrado al desplazamiento de la posición de las letras, haciendo corresponder una cifra a cada letra. En la práctica, cada permutación de la secuencia [0,1,2 ...,25] se llama clave y determina el cifrado del alfabeto. El problema es recordar la clave, que es una permutación del alfabeto. Sin embargo, hay sistemas de generación de permutaciones del alfabeto basadas en una letra, una palabra o una frase clave. Ilustramos estas afirmaciones con un ejemplo:

**Etapa 1**: elegimos una frase clave. Por ejemplo, una de las más conocidas y en inglés:

*to be or not to be that is the question.*

**Etapa 2**: quitamos los espacios entre las palabras y las letras que se repiten. Obtenemos:

*tobernhaisqu.*

**Etapa 3**: construimos nuestro alfabeto cifrado tomando, en orden, el alfabeto inglés (sin las letras ll y ñ) al completo, haciendo corresponder las letras a nuestra frase clave modificada, y completando con las letras del alfabeto que no están en la frase clave modificada en el orden original. Obtenemos:

a b c d e f g h i g k l m n o p q r s t u v w x y z

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

De esta manera tenemos una permutación del alfabeto asociada a nuestra frase clave. La tabla de cifrado es la siguiente:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

19 14 1 4 17 13 7 0 8 18 16 20 2 3 5 6 9 10 11 12 15 21 22 23 24 25

Si el mensaje que queremos enviar es el siguiente (hemos quitado los acentos para simplificar):

*No hay nada mas antiguo que la verdad*

aquí podemos ver el resultado después del cifrado:

*DF ATY DTET CTL TDMIHPF JPR UT VRKETE*

Con este método, la cantidad de claves que podemos generar aumenta sustancialmente, pasando de 25 a 26! (factorial), lo que corresponde al número posible de permutaciones de 26 letras del alfabeto. Esta cifra es aproximadamente 51·1018. Para entender la magnitud de este número, podemos compararlo con la edad del universo, que «solo» es de aproximadamente 15·109 años. En principio, este código parece indescifrable. Pero, ¿de verdad lo es?

Desgraciadamente, la respuesta es negativa. Porque la frecuencia de las letras en textos bastante largos y otros factores que dependen del tipo de alfabeto utilizado, pueden reducir sustancialmente la cantidad de intentos necesarios para encontrar la clave. Trataremos esta cuestión en los siguientes apartados.

Este tipo de código de cifrado se denomina «código monoalfabético». Hay otros códigos que utilizan varios alfabetos cifrados. Supongamos que utilizamos  alfabetos cifrados. Luego, dividimos el mensaje en bloques de *s* letras y ciframos las letras de cada bloque usando el alfabeto *s*, las utilizamos siempre en el mismo orden. Es decir, mediante *Ai* con  representamos al *iésimo* alfabeto, todas las letras en la *iésima* posición se codificarán con el *iésimo* alfabeto. Este tipo de cifrado se denomina «código polialfabético periódico». Cuando *s* es igual o superior a la longitud del mensaje, se dice que el código es «polialfabético aperiódico».

El primer ejemplo de este tipo de código, probablemente debemos agradecérselo a Leon Battista Alberti en el siglo XV. Alberti propone utilizar dos alfabetos cifrados para cada mensaje. Vigenère mejoró esta idea durante la segunda mitad del siglo XVI, proponiendo el uso de 26 alfabetos cifrados. En la figura 3.3 puede observar la tabla de Vigenère que representa los 26 alfabetos.

Imagen que contiene Texto

Descripción generada automáticamente

*Figura 3.3 - Tabla de Vigenère que representa los 26 alfabetos.*

Para simplificar, cada letra está asignada a un número entero, por ejemplo  y .

*Ejemplo*

Vamos a tomar la palabra clave "FISH" y el mensaje "shoot now".

La norma de codificación está contenida en la clave que comprende cuatro caracteres. Tenemos que utilizar el alfabeto de cifrado, que se repite cada cuatro letras (shoo | tnow), las líneas de la tabla de Vigenère corresponden a las letras F I S H. Así, la primera letra del mensaje, s, debe estar encriptada mediante el alfabeto correspondiente a la posición de la letra F, que es la posición 5 en la tabla de Vigenère, luego la segunda letra del mensaje, h, debe encriptarse mediante el alfabeto correspondiente a la posición de la letra I, y así sucesivamente.

La tabla 3.1 que aparece aquí debajo representa la frecuencia de aparición de las letras del alfabeto español.

|  |  |  |  |
| --- | --- | --- | --- |
| Letra | Frecuencia | Letra | Frecuencia |
| A | 12,53 % | Ñ | 0,31 % |
| B | 1,42 % | O | 8,68 % |
| C | 4,68 % | P | 2,51 % |
| D | 5,86 % | Q | 0,88 % |
| E | 13,68 % | R | 6,87 % |
| F | 0,69 % | S | 7,98 % |
| G | 1,01 % | T | 4,63 % |
| H | 0,70 % | U | 3,93 % |
| I | 6,25 % | V | 0,90 % |
| J | 0,44 % | W | 0,01 % |
| K | 0,02 % | X | 0,22 % |
| L | 4,97 % | Y | 0,90 % |
| M | 3,15 % | Z | 0,52 % |
| N | 6,71% |  |  |

*Tabla 3.1 - Frecuencia de aparición de las letras del alfabeto en español.*

*Fuente: Fletcher Pratt, Secret and Urgent the Story of Codes and Ciphers Blue Ribbon Books, 1939, pp. 254-255.*

Vamos a terminar este apartado con una última anécdota. En 1839, a través de las páginas del "Philadelphia Periodical", Edgar Allan Poe pidió a sus lectores que propusieran criptogramas utilizando códigos de sustitución monoalfabéticos y garantizó que los resolvería. Un día, recibió uno y dedujo que eran símbolos generados aleatoriamente. En 1975, el matemático Bryan J. Winkel y el investigador en química Mark Lister, que asistió a los cursos de criptografía de Winkel, descifraron el mensaje que había recibido Allan Poe.

Los dos hombres dedujeron que el mensaje no se había generado de manera aleatoria, pero tampoco se había codificado en monoalfabético. El mensaje contenía algunos errores, que probablemente se debían a faltas de transcripción.

## **2. Análisis de textos cifrados**

En los apartados anteriores hemos descrito algunas técnicas de cifrado de mensajes, que a primera vista eran indestructibles, si la única manera de procesarlas es la clasificación y el análisis de los errores. No era muy productivo. ¿Este proceso está completo o hay otra información que podríamos utilizar?

¿Y si abordamos la cuestión desde el punto de vista del adversario, intentando quebrar el mensaje encriptado a cualquier precio? Sabiendo que nuestro mensaje está encriptado por sustitución monoalfabética, para resolver este problema en un tiempo "admisible" o razonable no podemos proceder con el sistema de clasificación y detección de errores. Por lo tanto, tenemos que utilizar otros métodos independientes del tipo de clave utilizada para encriptar el mensaje. ¿Cómo proceder si solo tenemos un texto cifrado? Se trata de un problema vinculado al análisis de textos.

Se pueden tener en cuenta las propiedades del idioma en función del que se haya utilizado para redactar el mensaje. En un idioma con el italiano, la mayoría de las palabras terminan con la vocal «a». Por lo tanto, todos los símbolos situados al final de las palabras del texto cifrado son vocales. En general, un análisis estático puede proporcionar mucha información. Hay letras que aparecen con mucha frecuencia en algunos idiomas, mientras que en otros rara vez se utilizan. Partiendo de estos datos lingüísticos y estáticos, es posible construir la tabla 3.2 que representa la frecuencia de cada letra en español y en otros idiomas.

Tabla

Descripción generada automáticamente

*Tabla 3.2 - Diagrama comparativo de la frecuencia de las letras entre distintos idiomas. Fuente:*[*https://fr.wikipedia.org/wiki/Fr%C3%A9quence\_d%27apparition\_des\_lettres\_en\_fran%C3%A7ais*](https://fr.wikipedia.org/wiki/Fr%C3%A9quence_d%27apparition_des_lettres_en_fran%C3%A7ais)

Si tenemos un texto cifrado con un mensaje bastante largo escrito en español, entonces podemos determinar la frecuencia de las letras. El análisis de la frecuencia permite encontrar paso a paso las distintas letras en función de su porcentaje de aparición en el texto. Si obtenemos una palabra sin sentido, solo tenemos que ajustar la estrategia de asignación. La frecuencia de las letras dobles también puede darnos otra información, porque es muy frecuente que algunas letras aparezcan juntas.

Los métodos de criptoanálisis anteriormente mencionados se pueden aplicar a las antiguas inscripciones y a la arqueología en particular. Obviamente, los antiguos pueblos que se encuentran en el origen de estas inscripciones no eran criptógrafos y su objetivo no era encriptar los mensajes. Pero, para nosotros, estos textos son algo así como mensajes cifrados que tenemos que desencriptar. Desencriptar estos símbolos desconocidos tiene algo de magia, porque eso nos permite sumergirnos en el mundo antiguo, en las antiguas civilizaciones y retroceder en el tiempo. El ejemplo más elocuente son los jeroglíficos egipcios: el interés por restos símbolos egipcios se remonta al siglo XVI y lo inició el papa Sixto V. Pero el descubrimiento arqueológico más conocido es sin duda la piedra Rosetta, descubierta por Champollion en 1799 cerca del delta del Nilo. Los textos de oraciones en honor de los faraones, grabados en basalto negro, estaban escritos en tres idiomas: jeroglíficos egipcios, egipcio demótico y griego. La traducción del texto en griego permitió a Champollion desencriptar estos símbolos en 1822. Actualmente, la piedra Rosetta está expuesta en el British Museum, en Londres.

Para completar nuestra descripción histórica, ahora vamos a hablar de las distintas máquinas, desarrolladas a lo largo de la historia, para aplicar estos códigos de cifrado.

## **3. Máquinas de encriptado**

La primera máquina de cifrado de textos es el disco de cifrado de Leon Battista Alberti, estaba compuesta de dos discos de cifrado concéntricos de distintos diámetros, uno giraba sobre el otro en torno a un eje central. A lo largo de la circunferencia de los discos, hay dos alfabetos grabados. El desfase entre los dos discos remite a la clave *n*, y las correspondencias entre los dos permiten encriptar y desencriptar textos. En la figura 3.4 se puede observar una ilustración de estos dos discos.

Imagen que contiene Círculo

Descripción generada automáticamente

*Figura 3.4 - Disco de cifrado de Leon Battista Alberti. Fuente: Wikipedia (*[*https://es.wikipedia.org/wiki/Cifrado\_de\_Alberti*](https://es.wikipedia.org/wiki/Cifrado_de_Alberti)*).*

Esta máquina se utilizó desde el siglo XV hasta el final de Primera Guerra Mundial. Luego fue sustituida por la máquina Enigma, inventada y construida por Arthur Sacheries y Richard Ritter. Durante la Segunda Guerra Mundial, los alemanes utilizaron mucho esta última para encriptar sus comunicaciones. Inicialmente, estaba formada por tres elementos principales conectados entre ellos por hilos eléctricos (figura 3.5):

* una teclado alfabético para introducir el texto a cifrar;
* una unidad de interferencia que permitía cifrar el mensaje;
* un cuadro de indicadores con pequeños pilotos luminosos que tenían el mismo nombre que las letras del alfabeto, organizados de forma que el circuito eléctrico ilumina el piloto correspondiente a la letra encriptada.

Un teclado de computadora

Descripción generada automáticamente con confianza media

*Figura 3.5 - La máquina Enigma y sus cilindros. Fuente: Wikipedia (*[*https://es.wikipedia.org/wiki/Enigma\_(m%C3%A1quina)*](https://es.wikipedia.org/wiki/Enigma_(m%C3%A1quina))*).*

La unidad de interferencia (o simplemente el cilindro) es la parte central de la máquina. Consiste en una cinta gruesa en forma de disco atravesado por una red eléctrica compleja. Si queremos, por ejemplo, encriptar la letra "a" como "D", se introduce "a" en la máquina mediante el teclado, lo que desencadena una corriente eléctrica que pasa por el distorsionador y se enciende el piloto correspondiente a la letra D. La máquina ha sufrido varias mejoras, especialmente con la automatización del giro del disco de interferencia después de cada letra, y luego añadiendo otros dos cilindros.

Hasta 1943 los alemanes pensaban que Enigma era indestructible, pero los ingleses liderados por el matemático Alan Turing elaboraron calculadoras enormes capaces de desencriptar los mensajes de Enigma (la máquina de Turing se basa en un análisis estadístico), lo que permitió invertir la relación de fuerzas entre los alemanes y los aliados. El enorme esfuerzo de los ingleses por derrotar a Enigma contribuyó al auge de disciplinas nuevas, como la informática y la inteligencia artificial.

## **4. Contexto matemático de un criptosistema**

En el ámbito del secreto hay dos grupos: los criptólogos, que elaboran métodos y estrategias para disimular el sentido de los mensajes a las personas no autorizadas a acceder a ellos, y los criptoanalistas, que tienen el objetivo de desencriptar estos mensajes utilizando los fallos de los sistemas criptográficos. La interacción entre estos dos grupos, que tienen dos puntos de vista distintos, permite la aparición de sistemas criptográficos muy complejos y muy protegidos.

En este apartado vamos a describir el contexto matemático de la criptografía (criptología y criptoanálisis).

Antes de empezar, es necesario hacer algunas observaciones.

* El alfabeto utilizado en el mensaje encriptado y el mensaje desencriptado puede ser distinto. En general, es más interesante utilizar números en lugar de letras porque entonces es más sencillo describir los métodos de encriptado y de desencriptado.
* El procedimiento de transformación, que es la función que describe el paso de un texto no cifrado a un texto cifrado, debe ser biyectivo, para ser capaz de invertir el procedimiento, de desencriptar el mensaje y de tener el texto original. El único aspecto fundamental es que es necesario tener una clave para poder desencriptar el mensaje.
* ¿Por qué necesitamos una clave? Después de haber desarrollado un método seguro para enviar mensajes encriptados, cambiar de claves con frecuencia proporciona una seguridad mejor sin tener que cambiar todo el método o todo el sistema de encriptado. Podríamos esquematizar este dispositivo mediante un sistema de clave-cerradura.
* La clave debe tener un tamaño determinado para garantizar la seguridad del sistema.

En conclusión, un criptosistema consiste en:

* un conjunto *P* que contiene todos los textos posibles para cifrar.  representa un elemento de *P*;
* un conjunto *K* que contiene todas las claves posibles. Cada elemento  determina una transformación de encriptado  y su inverso de desencriptado . En concreto, ;
* un conjunto *C* que contiene todos los mensajes encriptados. *c* es un elemento de CM.

Dado un criptosistema determinado por (*P*,*C*,*K*), con:

*P* = [*textos no cifrados*],    *C* = [*textos cifrados*],     *K* = [*claves*]

La figura 3.6 describe la comunicación entre dos personas, James Bond y Moneypenny:

Diagrama

Descripción generada automáticamente

*Figura 3.6 - Esquema de comunicación encriptada entre dos personas (dos espías).*

En general, el texto no cifrado y el texto encriptado están divididos en unidades elementales, los bloques. Estos bloques pueden estar formados por un elemento (una letra), dos elementos (dos letras) o s letras. La principal ventaja de esta subdivisión es evitar reconocer el principio y el final de una palabra para hacer que el criptoanálisis basado en las frecuencias de las letras sea más complicado.

Cuando un mensaje unitario está formado por varias letras, debemos hacer corresponder cada letra con un número compuesto por varias cifras (por ejemplo, dos) o con su equivalente binario para evitar ambigüedades. En la figura 3.7 podemos ver la equivalencia entre las letras del alfabeto y los números de dos cifras decimales, así como su representación binaria.

Tabla

Descripción generada automáticamente

*Figura 3.7 - Tabla de equivalencia entre las letras del alfabeto y los números de dos cifras, así como su representación binaria.*

Nos damos cuenta de que al utilizar el sistema binario hacemos desaparecer cualquier ambigüedad sobre las letras correspondientes. Vamos a ver el ejemplo de dos mensajes con solo dos letras cada uno, «le» y «bo», que tienen el mismo equivalente numérico, pero su representación es distinta en dos cifras y en binario. Esta situación se resume en la tabla 3.3:

| **Columna1** | **Equi. Numérico** | **2 cifras** | **Binario** |
| --- | --- | --- | --- |
| le | 114 | 1104 | 0101100100 |
| bo | 114 | 0114 | 0000101110 |

*Tabla 3.3 - Ejemplo de ambigüedad entre las letras del alfabeto y el equivalente numérico de una cifra decimal.*

Otra manera estándar de asociar cada letra del alfabeto y cada carácter a un número es utilizar la nomenclatura ASCII (*American Standard Code for Information Interchange*). En la tabla 3.4 podemos ver la descripción de este estándar:

Tabla

Descripción generada automáticamente

*Tabla 3.4 - Estándar ASCII de asociación de cada letra del alfabeto y cada carácter con un número. Fuente:*[*http://www.asciitable.com/*](http://www.asciitable.com/)

## **5. Aritmética modular aplicada a algunos códigos de cifrado**

Recordemos que la aritmética modular agrupa a un conjunto de métodos de la teoría del álgebra de los números, lo que permite resolver problemas con números enteros. Esto deriva del estudio de los “restos” obtenidos por división euclidiana.

Ahora vamos a describir el aspecto matemático de algunos códigos de encriptado ya mencionados. También veremos un ejemplo para ilustrar mejor nuestras afirmaciones.

Dada la longitud del alfabeto *N* (generalmente *N* = 26). Una vez determinada la longitud *s* del tamaño de los bloques unitarios, si llamamos  a los equivalentes numéricos de cada letra, tenemos que describir el procedimiento que permite escribir cada bloque . También debemos asociarle a cada bloque un número  que en la base *N* es (). Sabemos que , entonces está claro que un cifrado es una función irreversible que toma sus valores en .

En el ejemplo anterior, donde *N* = 26, teníamos *le* = 114 y *bo* = 114. En este caso, tendremos:

 y .

Y a la inversa, si empezamos por el número 40 y sabiendo que *N* = 26, entonces podemos determinar que el texto no cifrado correspondiente es «*bo*» y solo «*bo*». En cierto modo, hemos resuelto una forma de ambigüedad. Otra alternativa para definir un bloque sin ninguna ambigüedad, es representarlo como un elemento de , donde cada  pertenece a . Entonces tenemos una biyección evidente entre  y  dada por:





En general, si dividimos un mensaje en bloques de s letras, tomando *N* letras del alfabeto, entonces la función de encriptado es una biyección:



De ahora en adelante, el procedimiento de cifrado que vamos a describir se denomina clásico, porque la clave de desencriptado se calcula fácilmente mediante la clave de encriptado. Es decir, el conocimiento de la clave de desencriptado es en esencia equivalente al conocimiento de la clave de encriptado (en algunos casos son idénticas). En el encriptado con clave pública, la divulgación de la clave de cifrado no compromete en nada la integridad de la clave de desencriptado. El primer tipo de encriptado se denomina simétrico o «two-ways», y el segundo se denomina asimétrico o «one-way».

Primero vamos a ver los códigos de encriptado clásico o simétrico. Estos códigos se basan en aritmética modular. Para simplificar, las equivalencias numéricas utilizadas en los ejemplos que vamos a exponer presentan espacios entre los números para evitar ambigüedades.

Cifrado o criptografía afín

Dado un mensaje unitario representado por una sola letra. Lo que significa que la representación numérica del mensaje es . Si *P* es una letra del alfabeto, también debe definir su equivalente numérico. *C* representa el criptograma de *P*.

Un encriptado afín se describe mediante una función de cifrado, que utiliza una función afín y es una biyección:



donde , el par  es la clave del sistema y *mod* es el módulo. Para que *C* sea una función biyectiva, es necesario que  sea primo relativo con 26, lo que da . En este caso, la congruencia  presenta una solución única  módulo 26. Por lo tanto, la función de desencriptado  es:



Consideramos . No hay que olvidar que el encriptado por translación es una función afín para la que . Obviamente, podemos escribir  y , tomando  y  módulo 26.

Recordemos que  es primo con  si su MCD es igual a 1.

En esta configuración, ¿de cuántas cifras afines disponemos? ¿De cuántos elementos en la ecuación que aparece a continuación? Es decir, la cantidad de pares de claves.



De todo ello deducimos que existe 12 · 26 = 312 pares de claves distintas, incluyendo la clave trivial (1,0).

Ahora tomamos el ejemplo de un código de encriptado con una clave (7,10). Según la identidad de Bézout, para los números 7 y 26, tenemos la siguiente ecuación:

-11 · 7 + 3 · 26 = 1

En este caso,  y la función de desencriptado es:



Lo que da: , porque .

La figura 3.8 resume el procedimiento de encriptado y desencriptado mediante una función afín.

Diagrama

Descripción generada automáticamente

*Figura 3.8 - Esquema que resume el procedimiento de encriptado y de desencriptado mediante una función afín.*

Retomamos el ejemplo anterior en el que James Bond envía un mensaje a Moneypenny.

**Criptoanálisis**: ahora, intentamos aplicar el análisis frecuencial a un sistema interceptado, sabiendo que este mensaje está en español, para determinar los coeficientes  y .

En el texto encriptado aparecen dos letras con una frecuencia baja, su incidencia es R y S, después de examinar el porcentaje de incidencia deducimos que corresponden a las letras q y z en el mensaje no cifrado. Según la relación afín anterior , podemos construir el siguiente sistema:



Este sistema de ecuaciones se puede reescribir en forma matricial. Obtenemos:

, con .

Para resolver este sistema de ecuaciones, tenemos que calcular la inversa de la matriz *A*.

Calculamos el determinante de , y obtenemos:



la inversa del módulo 26 de la matriz *A*. Entonces, la solución de nuestro sistema de ecuaciones es:



La solución que acabamos de describir representa un método genérico para el desencriptado mediante análisis frecuencial.

Es importante hacer algunas reservas sobre el método de criptoanálisis que hemos expuesto antes, para encontrar los parámetros  y . El sistema lineal de ecuaciones descrito admite una solución si y solo si A es tiene una inversa con módulo m. El sistema también puede admitir muchas soluciones, y en este caso, es imperativo probarlas todas.

En algunos casos, no existe la inversa del determinante de *A* con módulo *m*, sino con módulo un número primo *q* divisor de *m*. Así podemos resolver el problema con módulo *q* y luego tendremos que analizar este resultado para determinar si es adecuado.

Cifrado o encriptado de Hill

En este tipo de encriptado, el mensaje está subdividido en bloques de longitud *s* y, para cada bloque, aplicamos una función de cifrado así:



donde *A* es una matriz cuadrada *s* x *s*, *b* es un vector columna de longitud *s*, *p* y *c* son los vectores columna de longitud *s* correspondientes al texto no cifrado y al texto cifrado. Para poder descifrar el modelo, es necesario que se cumpla la condición de inversión módulo 26 en la matriz A.



Este sistema es la transformación afín definida por la clave .

En el ejemplo que aparece a continuación, vamos a aplicar el sistema de ecuaciones para una transformación definida por la clave:



En este caso, la función de encriptado es:



Podemos observar que *A* tiene una inversa módulo 26, . Entonces calculamos la inversa módulo 26 de la matriz *A*. La inversa de 21 módulo 26 es 5 porque la identidad de Bézout (Goldschmidt, 1964) es:

5 · 21 - 4 · 26 = 1

Entonces, la inversa módulo 26 de *A* es:



Y la función de desencriptado es:



# Criptografía de clave pública: fundamentos

En este apartado vamos a proseguir con la descripción de los sistemas de criptografía centrándonos en los sistemas de clave pública. Posteriormente, ilustraremos nuestras afirmaciones con un sistema criptográfico específico basado en el problema de "Knapsack", el sistema RSA (Adleman, 1978).

En los códigos de cifrado que hemos descrito hasta ahora, el procedimiento de desencriptado no es difícil si se conoce el método y la clave de encriptado. En estos casos, hay una simetría entre las funciones de encriptado y de desencriptado: son del mismo tipo desde el punto de vista lógico y de cálculo.

En la época actual donde toda la información se transmite por teléfono, correo electrónico o radio, todos los mensajes, incluidas las claves, son susceptibles de ser interceptados. Por eso es muy importante permitir que los usuarios se comuniquen de manera segura sin tener que intercambiar claves de encriptado. La criptografía de clave pública se desarrolló con este propósito.

La criptografía de clave pública permite hacer públicos el método y la clave de cifrado sin revelar ninguna información sobre el desencriptado de los mensajes. Es decir, en este sistema, para poder desencriptar el mensaje en un tiempo razonable hay que tener información complementaria además de la clave pública. Esta información se mantiene en secreto, y sin ella la complejidad de desencriptado es bastante elevada para hacer que el desencriptado sea imposible. Descifrar el mensaje sin la información complementaria necesitaría un tiempo muy superior al que se necesita para el cifrado.

El ejemplo de la guía telefónica de una gran ciudad ilustra muy bien la diferencia entre ser capaz de hacer una operación y ser capaz de realizar la operación inversa. Porque es fácil encontrar un número de teléfono a partir de un nombre, pero el proceso inverso consistente en encontrar un nombre a partir de un número es francamente imposible, o al menos se necesitaría una cantidad de tiempo extraordinaria.

Desde un punto de vista matemático, esta idea nos remite al concepto de función de sentido único. Una función de sentido único  es una función que se puede calcular fácilmente (en tiempo polinómico), pero eligiendo  de manera aleatoria, es más difícil (tiempo exponencial) calcular  tal que . Este concepto puede parecer un poco vago, porque los términos «fácil» o «aleatorio» no tienen definiciones matemáticas rigurosas, pero puede darnos una idea clara de las funciones de sentido único.

Para ilustrar nuestra afirmaciones, vamos a examinar el siguiente ejemplo.

Consideramos un grupo finito *G* de orden *n* y un elemento .

Sea:



Se considera la siguiente función exponencial:



Si , *x* es el logaritmo discreto de *y* en *G* con la base , y lo representamos como . Cuando *G* es el grupo multiplicativo  de campo final , si  es uno de sus generadores, entonces  es biyectiva y su función inversa se llama logaritmo discreto en , con base . En este caso, el cálculo de  es de tiempo polinómico. Por otro lado, todo los algoritmos conocidos de cálculo de logaritmos discretos son de tiempo exponencial. Por lo tanto, las funciones exponenciales se pueden considerar como funciones de sentido único. Observamos que algunos logaritmos para el cálculo de logaritmos discretos, como «baby step-giant step», en algunos casos son especialmente eficaces.

En general, en criptografía, es interesante considerar este grupo de funciones, porque el cálculo de funciones exponenciales es relativamente sencillo, por el contrario, el cálculo de logaritmos discretos es más difícil.

Hay otros tipos de funciones  más fáciles de calcular que sus inversas , pero, al contrario que con las funciones exponenciales y logarítmicas, las dos se calculan en tiempo polinómico. Estos tipos de funciones se pueden representar mediante algoritmos de multiplicación utilizando matrices cuadradas , porque el cálculo de  requiere aproximadamente *s*2 operaciones, mientras que el cálculo de la función inversa requiere más operaciones, aproximadamente *s*3, por la inversión de la matriz.

Como conclusión sobre las funciones de sentido único, su existencia no se ha demostrado con rigor. Sin embargo, hay serios candidatos, como las funciones exponenciales que tienen las características adecuadas.

## **1. Algoritmo para calcular logaritmos discretos**

Vamos a describir el famoso algoritmo «baby step-giant step», que permite calcular los logaritmos discretos en un campo , siendo  un número primo. Trabajaremos en  cuyos elementos son: . Sea  un generador de . Para determinar el logaritmo discreto  de  en la base , procederemos de la siguiente manera:

* baby steps: fijamos  como el número entero más pequeño superior a , calculamos los valores  para todo  y los insertamos en una lista para guardarlos en la memoria.
* giant steps: calculamos , luego  y sucesivamente . Después de cada cálculo, comparamos el resultado con la lista creada en la primera etapa, y desde que obtenemos una igualdad de la forma , encontramos el logaritmo de . Hay que señalar que el algoritmo encuentra el logaritmo exacto deseado.

El logaritmo es un número que se encuentra en . Entonces, dividiendo  por , tenemos  con  y, como , también tenemos .

La complejidad de este algoritmo es la siguiente: en la primera etapa, tenemos  operaciones, y cada una de ellas tiene una complejidad el orden de , en total tenemos una complejidad de . También hay que destacar que esta etapa se calcula una sola vez y es independiente del número *y* del que queremos calcular el logaritmo. Razonando de la misma manera, encontramos que la complejidad de la segunda etapa también es del orden de .

Hay que tener otras aspectos en cuenta, como las necesidades de memoria, porque hay que guardar en memoria la lista de la primera etapa. También tenemos la comparación entre los números de la etapa 1 y de la etapa 2. La complejidad de esta etapa de comparación es del orden de .

En total, sumando las complejidades, podemos afirmar que el algoritmo tiene una complejidad exponencial.

*Ejemplo*

Estudiamos un ejemplo sencillo. Tomamos  y , que es el generador de . En este caso, tenemos , entonces . La etapa baby steps da:

20 = 1, 21 = 2, 22 = 4, 23 = 8

El cálculo de 24 es igual a 5 módulo 11 cuando su inversa es 9.

Supongamos que deseamos calcular el logaritmo . En este sentido, ejecutamos la etapa giant steps.

Al principio, calculamos 6 · 2-4, que es igual a 1 módulo 11. Este número no está en la lista de la etapa baby steps, entonces este no es el logaritmo correcto. Calculamos otra etapa, 6 · 2-8, que es igual a 2 módulo 11. Entonces hemos encontrado la relación 6 · 2-8 = 2 o 6 = 29 (*mod*11). Entonces el resultado es .

Puede encontrar otros algoritmos para calcular logaritmos discretos en (Koblitz, 1994).

## **2. Problema de la mochila y su aplicación a la criptografía**

Supongamos que vamos a salir de excursión. Tenemos que preparar la mochila optimizando al máximo el espacio disponible en el interior. Disponemos de un número  de objetos con los volúmenes  y sabemos que la mochila tiene un volumen *V*. Deseamos transportar el máximo de carga en la mochila. ¿Cómo alcanzar este objetivo? La situación se puede esquematizar el subconjunto  de tal manera que:



Este esquema se puede aplicar a muchos problemas distintos. Por ejemplo, tenemos que pagar 2 euros y disponemos de monedas de 2, 5 y 10 céntimos: ¿cuál es la mejor combinación para pagar con la mínima cantidad de monedas? ¿O con el máximo de monedas? O incluso, ¿cuántas maneras distintas tenemos para pagar?

Volvamos a nuestro problema de la mochila y describamos la aplicación criptográfica detrás de este problema propuesto por (Hellman, 1978).

Primero, reformulamos el problema de la siguiente manera: para  enteros positivos  y otro número entero positivo *m*, podemos encontrar  enteros  con  tal que el número entero *m* se puede escribir:



En ciertos casos, la solución a este problema no existe y, en otros casos, no hay una solución única. Los tres ejemplos siguientes ilustran las tres situaciones posibles.

**Ejemplo 1**: sea  y *m*=21. Hay dos soluciones inmediatas: 21 = 2 + 8 + 11 et 21 = 2 + 7 + 12. Son las dos soluciones únicas a este problema. En la forma explícita, para la primera solución tenemos: , en cuanto a la segunda solución: .

**Ejemplo 2**: consideramos todo conjunto  y *m* = 1. Como m es inferior a todos los elementos , entonces es imposible escribir *m* como una combinación de  con coeficientes 0 o 1.

**Ejemplo 3**: si , para , resolver el problema de la mochila se reduce a escribir la representación binaria del número entero *m*. Y como sabemos, la solución es única.

Para encontrar la solución de este tipo de problemas, basta con considerar todas las sumas posibles, y existen  subconjuntos posibles incluyendo el conjunto vacío. Cuando  es pequeño, el problema se puede resolver, pero en caso contrario la resolución no es posible porque está en tiempo exponencial. En general, no hay ningún algoritmo que permita resolver el problema de la mochila, aparte de probar todas las posibilidades.

La dificultad del problema de la mochila es que está en la categoría de los algoritmos muy difíciles, llamados problemas *NP*, para los que no existen algoritmos que puedan resolverlos en tiempo polinómico. Se dice que un problema *P* es *NP* si existen algoritmos, no necesariamente polinómicos, que puedan resolverlo y si es posible verificar para ciertos datos, si este problema se puede resolver con un algoritmo polinómico determinista. Por el momento, el problema de factorización de un número entero forma parte de esta categoría.

La demostración de la complejidad del problema de la mochila está disponible en (Garey, Johnson, 1979).

Un caso particular del problema de la mochila es cuando la secuencia  es supercreciente en el sentido de la definición siguiente:

Se dice que una secuencia de  enteros positivos es supercreciente, si se verifican las siguientes desigualdades:

Texto, Carta

Descripción generada automáticamente

Dicho de otro modo, cada elemento de la secuencia es mayor que la suma de todos los elementos anteriores. ¿Entonces se plantea la pregunta de la existencia de solución en este caso? La respuesta es: no siempre, pero existe solución. Es única y se puede determinar en tiempo polinómico.

Para determinar , observamos que:



Luego, para determinar , sustituimos *m* por . Sabiendo que la secuencia  también es supercreciente, tenemos:



Al generalizar la condición anterior, obtenemos:

Esquemático

Descripción generada automáticamente con confianza media

Es obvio que si , hemos encontrado la solución y esta solución es única. Y si  pero  es mayor que , entonces no existe ninguna solución.

## **3. Cifrado con clave pública basado en el problema de la mochila, o cifrado Merkle-Hellman**

El cifrado se puede construir siguiendo las etapas que se describen a continuación:

Cada usuario *X* elige una secuencia súper creciente  de longitud fija N, un número entero m tal que , y un número entero  primo relativo con m. Estas informaciones se guardan en secreto.

El usuario *X* calcula la siguiente transformación:

, para .

*X* hace pública la secuencia  y representa la clave de cifrado.

Un usuario *Y* envía un mensaje  a *X* de la siguiente manera: primero, transforma cada letra del texto en el equivalente binario. Luego, subdivide la secuencia de unos y ceros en bloques de longitud *N*, añadiendo unos al final para completar el último bloque si es necesario, porque todos los bloques deben tener la misma longitud *N*. Para cada bloque, , el usuario *Y* aplica la transformación para encriptar , y envía el mensaje encriptado a *X*.

¿Cómo va a desencriptar *X* el mensaje de Y?

Primero, *X* calcula la clave de desencriptado que es  con .

Luego, calcula .

Considerando la definición de , tenemos:



Y como  es una secuencia súper creciente, tenemos:



Y entonces se obtiene .

*X* conoce el número entero  y la secuencia supercreciente ; a partir de esta información, debe construir los números enteros . Esto se puede realizar fácilmente en tiempo polinómico utilizando el algoritmo de la mochila para las secuencias súper crecientes.

El desencriptado de mensajes de manera ilegítima es un problema muy difícil de solucionar. Sin embargo, en 1982, Shamir (Shamir, 1982) encontró un algoritmo polinómico para desencriptar los mensajes cifrados mediante este método. Por lo tanto, no es un método seguro, pero hay variantes de este método que siguen resistiendo a los criptoanalistas.

La figura 3.9 representa un esquema de cifrado basado en el problema de la mochila.

Diagrama

Descripción generada automáticamente

*Figura 3.9 - Esquema de cifrado basado en el problema de la mochila.*

# Sistema RSA

Se trata de un sistema de criptografía desarrollado por W. Diffie y M. E. Hellman (Diffe, 1976), pero comúnmente llamado sistema RSA en virtud de los nombres de las personas que lo implementaron: Rivest, Shamir, Adleman, en el MIT (*Massachusetts Institute of Technology*) (Adleman, 1978).

Este sistema utiliza una clave pública para encriptar los datos. Cada usuario divulga su clave de cifrado y todos pueden comunicarse de manera segura. El algoritmo de encriptado utiliza una función con una trampa, porque el usuario conserva una parte de la información en secreto, que le permite desencriptar los mensajes invirtiendo la función de encriptado.

Las aplicaciones de este sistema son diversas: envío de mensajes secretos, autentificación electrónica, acceso protegido a las bases de datos, algunos servicios como el pago con tarjeta de crédito y el pago en internet, y muchos más. Este sistema en una red blockchain también permite responder a problemáticas de seguridad, de autentificación y de gestión de accesos.

## **1. Descripción del sistema RSA**

Vamos a utilizar el sistema RSA para intercambiar mensajes que solo pueden leer sus destinatarios. Entonces nos unimos al sistema, divulgamos la clave de encriptado. La clave de encriptado es un par de números enteros positivos , donde *n* es el producto de números grandes primos *p* y *q* que solo conocemos nosotros, y *e* debe ser primo relativo con , que es  o *MCD(e, p-1)=MCD(e,q-1)=1*. Este par de números enteros se almacena en una carpeta pública.

En la práctica, para encontrar un número primo suficientemente grande (con al menos 100 dígitos), utilizamos un generador aleatorio para generar un número impar m al que le aplicamos un test de primalidad para verificar si es un número primo. Si no es el caso, aplicamos la prueba a m+2, luego a m+4, hasta encontrar un número primo. Recordemos que la cantidad de números primos inferiores a m es m/log m.

Volvemos al sistema RSA. Cada usuario U va s seguir el mismo procedimiento, divulgando la clave pública  con las mismas características:  es el producto de dos números primos  y  que deben guardarse en secreto. Y hay que elegir  de manera que *MCD*(*eu*,*pu*-1) = *MCD*(*eu*,*qu*-1) = 1.

Para comprender mejor el funcionamiento de este sistema, vamos a ver un ejemplo.

*Ejemplo:*

En el directorio público, encontramos la clave de encriptado correspondiente al lado de cada nombre.

|  |  |
| --- | --- |
| Billel "A" | *A* = (247,5) |
| Federico "B" | *B* = (1003,3) |
| Juan Miguel "C" | *C* = (77,13) |
| Yong "D" | *D* = (703,7) |

Los números se han elegido en función de las condiciones requeridas, de la siguiente manera:

|  |  |
| --- | --- |
| n*A* = 247 = 13 . 19 | *MCD*(5,12) = *MCD*(5,18) = 1 |
| n*B* = 1003 = 17 . 59 | *MCD*(3,16) = *MCD*(3,58) = 1 |
| n*C* = 77 = 7 . 11 | *MCD*(13,6) = *MCD*(13,10) = 1 |
| n*D* = 703 = 19 . 37 | *MCD*(7,18) = *MCD*(7,36) = 1 |

En nuestro ejemplo, hemos elegido números pequeños para los que es fácil encontrar los dos números primos. En la realidad, el usuario elige números muy grandes con más de 100 dígitos. La manipulación de números primos muy grandes en este tipo de cifrado ha dinamizado la investigación en el campo de las pruebas de primalidad, que son primordiales.

## **2. Envío de un mensaje encriptado con el sistema RSA**

El usuario Juan Miguel ha recibido un mensaje de Federico con el texto:

¿Cuál es tu clase preferida?

Le gustaría responder: 

Para enviar esta respuesta a Federico, debe proceder de la siguiente manera:

Primero, tiene que transformar cada letra en su equivalente numérico de dos cifras. Lo que da:

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

Luego, tiene que buscar en el directorio público la clave de encriptado correspondiente a Federico, que es: 

Ahora, tiene que subdividir el mensaje en bloques unitarios, de tal manera que cada bloque sea más pequeño que  y primo relativo con 1003. Lo subdivide en bloques de dos letras, como se puede ver a continuación:



Los números correspondientes son:



Estos números 11, 604, 117, 23 son menores que 1003 y son primos relativos con 1003. Para encontrar el MCD entre estos números y 1003, Juan Miguel utiliza un algoritmo euclidiano, porque no conoce la descomposición en factores primos de .

El último bloque se ha completado con la letra "x" para tener un bloque de dos letras. Si hubiera hecho la descomposición en bloques de tres letras, rápidamente se habría encontrado con números mayores que 1003.

Al final, tiene la siguiente descomposición:



Esta operación nos permite decir que, para cada mensaje , se cumplen las siguientes condiciones.



A partir de esta etapa, puede empezar la fase de encriptado del mensaje. Para ello, Juan Miguel eleva cada mensaje  a la potencia :



Por lo tanto, el mensaje encriptado es:

Texto

Descripción generada automáticamente

y Federico recibe el siguiente mensaje:



Una vez que Federico ha recibido el mensaje encriptado debe proceder a descifrarlo.

## **3. Desencriptado de un mensaje RSA**

Federico ha recibido el siguiente mensaje:



Tiene que descifrar el mensaje, y para cada bloque, tiene que volver a construir el mensaje inicial sabiendo que:



¿Cómo determinará Federico la función de desencriptado ? Como ya hemos mencionado antes, Federico tiene información adicional que le permitirá desencriptar el mensaje. Primero, debe determinar , con:



tal que  es la solución de la siguiente congruencia:



Esta solución existe y es única porque .

En este caso, , Federico sabe que  y, por lo tanto, . Entonces, la solución es .

Este número es la clave privada, solo la conoce Federico porque es el único que conoce la factorización de 1003.

Para empezar, Federico eleva los bloques de mensaje que ha recibido a la potencia , lo que da:



El exponente 619 es muy grande, así que Federico procede de la siguiente manera:

619 = (1001101011)2 = 512 + 64 + 32 + 8 + 2 + 1

Obtiene:



Las potencias se calculan fácilmente en función de la tabla que aparece aquí debajo (para C1 = 328) donde los factores multiplicadores se indican en negrita.

|  |  |
| --- | --- |
| **k** | **Ck1 mod 1003** |
| **1** | **328** |
| **2** | 3282 mod 1003 = **263** |
| **22=4** | 2632 mod 1  003 = 965 |
| **23 =8** | 9652 mod 1003 = **441** |
| **24 = 16** | 4412 mod 1003 = 902 |
| **25 = 32** | 9022 mod 1003 = **171** |
| **26 = 64** | 1712 mod 1003 = **154** |
| **27 = 128** | 1542 mod 1003 = 647 |
| **28 = 256** | 6472 mod 1003 = 358 |
| **29 = 512** | 3582 mod 1003 = **783** |

Entonces Federico obtiene:





Por lo tanto *P*1 = 11, lo que corresponde a la primera parte del mensaje original. Federico aplica el mismo método para los otros bloques, lo que le permite obtener:



Los equivalentes en cuatro cifras decimales son: 0011, 0604, 0117, 0023.

Haciendo corresponder una letra a cada número, se obtiene:

00 11 06 04 01 17 00 23

*a l g e b r a x*

## **4. ¿Por qué funciona este método de desencriptado?**

Primero, la congruencia  presenta exactamente una solución porque el coeficiente es tal que:



y también: 

Federico es el único que puede descifrar el mensaje de Juan Miguel porque es el único que conoce la función de Euler , como conoce los factores primos, . En efecto, estos dos números primos son muy grandes y su factorización cuesta mucho tiempo. Esta factorización solo la conoce Federico.

En realidad, el conocimiento de  es equivalente al conocimiento de los factores primos . Pero, ¿se puede conocer  sin conocer ? La respuesta es no: porque conociendo , se pueden reconstruir los dos factores primos en tiempo polinómico (Baldoni, Ciliberto, Cattaneo, 2009).

En el método de encriptado de la palabra "álgebra" anteriormente descrito, esta palabra se subdivide en bloques de dos letras, luego haciendo pruebas y errores, Federico verifica que el número correspondiente a cada bloque es inferior a  y primo relativo con . Sin embargo, hay un método más eficaz para subdividir la palabra que se desea cifrar. La idea es la siguiente, en lugar de dividir el mensaje de origen, dividimos su equivalente numérico en k cifras, con:



Esto permite garantizar que el número de cada bloque es inferior a , sin tener que examinar el mensaje.

La figura 3.10 resume las distintas etapas del sistema RSA.

Diagrama

Descripción generada automáticamente

*Figura 3.10 - Esquema de cifrado con el sistema RSA.*

## **5. Autentificación y firmas con el sistema RSA**

El sistema RSA permite resolver un problema importante en el sector de las telecomunicaciones: el problema de autentificación de firmas.

Cuando Federico recibe el mensaje de Juan Miguel, ¿cómo puede estar seguro que procede de Juan Miguel?

La autentificación se puede conseguir de la siguiente manera:

Juan Miguel escribe el mensaje *P*1 y añade al final su firma *F*; para autentificar la firma, después del mensaje *P*1, Juan Miguel añade el mensaje



donde  es su clave privada. Así Juan Miguel envía un mensaje *P* a Federico formado por *P*1 y *P*2. Luego solo queda elevarlos a la potencia  y reducirlos al módulo *nB*.

Cuando Federico reciba el mensaje, utilizará su clave privada  para descifrar el mensaje *P*1 y sabrá que el mensaje es de Juan Miguel porque tiene la firma *F*. ¿Pero es de verdad la firma de Juan Miguel? La parte *P*2 permitirá la autentificación. En efecto, está formada por caracteres indescifrables, que, sin embargo, contienen la prueba de la autenticidad de la firma.

Para verificar la firma, Federico procede de la siguiente manera:

No puede utilizar su clave privada  para descifrar *P*2 porque no es de ninguna utilidad en este caso, el mensaje de origen *F* se encriptó elevando a la potencia , y no a . En su lugar, Federico utiliza la clave pública  de Juan Miguel. De esta manera obtiene la firma *F* de origen, porque:



Esta firma debe ser auténtica porque solo Juan Miguel conoce su clave privada. Si lo que aparece no es la firma *F*, entonces el mensaje no es auténtico y no es válido.

*Ejemplo*

Juan Miguel ha publicado la siguiente clave pública . Como , la clave privada de Juan Miguel es *dc* = 37, porque , donde . Para autentificar la firma de Juan Miguel, que suponemos *F* = 5, que ha sido elevada a la potencia  módulo  para encriptarla:



Federico eleva esta firma encriptada a la potencia :



Así obtiene la firma de origen (no encriptada) de Juan Miguel. Entonces está seguro del origen y de la autenticidad del mensaje.

La figura 3.11 resume el procedimiento de autentificación de una firma electrónica con el sistema RSA.

Diagrama

Descripción generada automáticamente

*Figura 3.11 - Procedimiento de autentificación con el sistema RSA (firma electrónica).*

## **6. Seguridad del sistema RSA**

La seguridad del sistema RSA está vinculada al hecho de que actualmente es imposible tener un algoritmo eficaz que pueda factorizar un número muy grande. Si *C* envía un mensaje *A* a *B*, para poder desencriptarlo el pirata tiene que encontrar los dos números primos de más de 60 dígitos cada uno, que sirven para la factorización de . El cálculo de estos dos factores puede requerir un tiempo considerable, algunos meses o incluso algunos años, incluso con el ordenador más rápido.

El agosto de 1977, los tres inventores del sistema RSA propusieron un reto: ofrecieron 100 dólares a todo aquel que pudiera conseguir factorizar un número de 129 dígitos para desencriptar un mensaje secreto. Para los más curiosos, estas son las líneas generales del problema.

El número que se desea factorizar es:

*N*=114381625757888867669235779976146612010218296721242362562

561842935706935245733897830597123563958705058989075147599

290026879543541

Por otro lado, *e* = 9007 y (*N*,*e*) es la clave pública de Rivest, Shamir y Adleman. Para garantizar que el mensaje fue publicado por el equipo del MIT, se ha añadido esta firma al final del mensaje:

167178611503808442460152713891683982454369010323583112178350

3844 69290626554487922371144905095786086556624965779748

40004057020373

Elevando este número a la potencia 9007 y reduciéndolo al módulo N, obtenemos el siguiente número:

0609181920001915122205180023091419001

5140500082114041805040004151212011819

Este número corresponde a la firma: «First solver wins one hundred dollars» (el primero en resolverlo consigue cien dólares), lo que garantiza que el mensaje procede del equipo del MIT.

Rivest, Shamir y Adleman confiaban en su sistema y pensaban que era inviolable. Diecisiete años más tarde, el matemático holandés Arjen K. Lenstra, con la ayuda de un centenar de personas en todo el mundo, consiguió encontrar la solución en ocho meses utilizando el método de criba cuadrática. Este método permite subdividir una tarea en varias tareas más pequeñas.

El mensaje original enviado por los tres inventores es: «the magic words are squeamish ossifrage» (las palabras mágicas son quebrantahuesos aprensivo). Los tres científicos confesaron que la frase no tiene ningún sentido y no esperaban que un día saliera a la luz.

# Variantes del sistema RSA

De ahora en adelante, vamos a presentar algunos criptosistemas que son variantes del sistema RSA.

## **1. Intercambio de claves privadas**

Una ligera modificación del sistema RSA permite que dos usuarios intercambien una clave privada con la que pueden intercambiar mensajes independientemente de la clave pública utilizando métodos clásicos de cifrado.

El sistema RSA se puede modificar de la siguiente manera. Elegimos un número primo grande , que se divulgará para trabajar en un campo . Actualmente, podemos trabajar en cualquier campo finito , pero debemos limitarnos a un campo con  elementos. Luego, elegimos otros elemento no nulo , que también se divulgará. Cada usuario *U* elige su clave privada , un número positivo inferior a , y divulga  que es un número positivo tal que . Hay que destacar que es imposible reconstruir la clave privada  en un tiempo razonable a partir de , porque eso implica el uso de logaritmos discretos y, como ya hemos visto, eso requiere un tiempo exponencial.

Para intercambiar una clave privada, dos usuarios *A* y *B* pueden, por ejemplo, ponerse de acuerdo para utilizar la clave privada , tal que:



En efecto, *A* y *B* pueden calcular  en tiempo polinómico. *A* conoce  que es pública y también conoce su clave privada . Por lo tanto, puede calcular . De manera similar,  puede calcular .

El hecho de conocer  y  no permite reconstruir  en tiempo polinómico. La cuestión que puede plantearse es la siguiente: ¿es necesario conocer  y  para calcular  conociendo  y ? Hasta ahora, no se ha rechazado esta norma estricta. Sin embargo, sabemos que la complejidad de este problema es igual a la complejidad de encontrar un logaritmo discreto. Esta afirmación es la hipótesis de Diffie-Helleman y constituye la base de la seguridad de este método.

## **2. Prueba de divulgación de conocimiento cero**

Supongamos que Billel ha encontrado una fórmula muy importante. Le gustaría persuadir a sus colegas de que la encontrada, pero sin revelar indicaciones sobre la fórmula o manera de demostrarla. ¿Es posible? Este tipo de comunicación se llama prueba de divulgación de conocimiento cero. Se trata de un tipo de comunicación que permite no transmitir ninguna información sobre la fórmula y la manera de demostrarla, pero da la prueba de posesión de la fórmula al destinatario del mensaje. Parece imposible, pero sin embargo es posible. Vamos a ilustrarlo con el siguiente ejemplo.

Dado un grupo finito *G* de *N* elementos, y dos elementos de *G*: *b* e *y*. Supongamos que Billel ha encontrado una fórmula de logaritmo discreto para *y* en la base *b*. Es decir, ha determinado el número entero positivo *x* tal que:



Su amigo Yong es escéptico: Billel quiere convencerlo de que conoce el número entero *x* sin divulgarlo. Entonces pueden proceder de la siguiente manera:

Billel genera un número entero positivo aleatorio *e* < *N* y se lo envía a Yong:



Yong lanza una moneda.

* Si sale cruz, Billel debe transmitirle *e* a Yong que verifica si .
* Si sale cara, Billel debe transmitirle el número entero positivo . Como  e , tenemos . Yong puede verificar si el número tiene la propiedad correcta (hay que señalar que Yong conoce ).

Las tres etapas se repetirán hasta que Yong esté convencido de que Billel ha encontrado realmente el logaritmo discreto de *y*.

Pero, ¿cómo convencer a Yong? Si Billel no conociera de verdad el logaritmo discreto de *y*, solo podría responder a una o dos preguntas. Porque, cuando se tira la moneda y sale cara, Billel debe transmitir , algo que solo puede hacer si conoce *x*. Billel puede hacer trampa, durante la etapa 1, enviando a Yong  en lugar de . En este caso, si al tirar la moneda sale cara, transmitirá  (algo fácil de realizar). Pero si sale cruz, debe revelar , y no puede hacerlo sin conocer *x*. Iterando este procedimiento, llegará un momento en que Yong se convencerá de que Billel realmente conoce *x*.

Por lo tanto, Billel ha conseguido demostrarle a Yong que conocía el logaritmo discreto de *y*, sin haber divulgado *x* de manera explícita.

## **3. Funciones de hash**

Dados dos conjuntos *X* y *M*. Una función de hash  es una función sobreyectiva que transforma un mensaje *m* de una longitud finita cualesquiera, en un mensaje *h* condensado de longitud fija  y que satisface las siguientes propiedades:

* 1. *H*(m) debe calcularse de manera fácil y rápida;
* 2. *H* debe ser una función de sentido único;
* 3. *H* no debe tener colisiones.

En términos probabilísticos, hay menos mensajes condensados que mensajes no transformados por una función hash, lo que induce una probabilidad no nula de tener el mismo condensado para dos mensajes distintos. Es decir, tendremos los mensajes *m* y *m’* tales que *H*(m)=*H*(m’). Pero esta probabilidad debe ser muy baja. También observaremos que las condiciones 1 y 3 compiten entre sí porque, cuando construimos una función de hash de tal manera que sea fácil de calcular, es legítimo pensar que además no presenta ninguna colisión. Por ejemplo, es posible no poner cotas o límites a la longitud del mensaje condensado (el condensado). Así, nos aseguramos de la independencia entre el mensaje en la entrada y el condensado. Sin embargo, el tiempo de tratamiento es proporcional a la longitud del mensaje al que se aplicará la función hash. En la práctica, nos aseguramos de la condición 3 simplemente pidiendo que para un mensaje dado *m* sea imposible encontrar un  tal que  y  en tiempo polinómico. Sin embargo, sigue siendo muy difícil hacer realidad esta característica cuando desarrollamos una función de hash. Entre las funciones de hash más utilizadas, podemos citar Secure Hash Algorithm (SHA-1) y la familia de Message Digest (MD). El primer algoritmo que se publicó fue MD2, diseñado en 1989 por Ronald Rivest. Al que siguieron las versiones mejoradas MD4 y MD5. El MD5, introducido en 1991 para sustituir al MD4, produce un mensaje condensado de 128 bits. Como podremos constatar un poco más adelante en este mismo capítulo, ahora sabemos que eso no es suficiente para garantizar una seguridad elevada. En 1996 se actualizó un fallo con el MD5, demostrando que había una posibilidad de encontrar colisiones en los mensajes tratados con una función hash. Así, el MD5 se retiró progresivamente para ser sustituido por el SHA-1.

En 2004, se constataron colisiones completas, demostrando así que el MD5 ya no es seguro desde el punto de vista criptográfico.

El algoritmo SHA-1 produce un mensaje de 160 bits al que se aplicado una función hash. El mensaje *m* de longitud *n*, está dividido en bloques de longitud *l*, donde el último bloque se ha completado, si era necesario, con una secuencia de bits, empezando por un 1 seguido únicamente por varios 0, hasta que la longitud del mensaje sea  bits, donde  es la función límite. La cantidad de bits del mensaje se completa con una representación binaria de la longitud del mensaje inicial (una secuencia de bits de 64 bits de longitud). Por ejemplo, para un mensaje de 2800 bits, obtenemos un mensaje de longitud: 3008 = 6 x 512 — 64. El mensaje final tendrá una longitud de 3072 después de añadir 64 bits.

A continuación, el mensaje se divide en *l* bloques de 512 bits (*m* = (*m* 1,*m* 2,...,*ml*)). Cada bloque  ve su secuencia modificada por diversas series de transformaciones utilizando funciones de compresión *h*. Empieza por un valor inicial *X*0 y definimos . El  final es el mensaje procesado por la función hash. La idea detrás de la construcción de las funciones de compresión es la siguiente: una fuerte dependencia del mensaje inicial. Es decir, el cambio de una sola unidad o de un solo bit del mensaje de origen afectará de manera segura a la mayor cantidad posible de bits en el condensado (el resultado después de aplicar la función hash).

Subsisten diferencias entre las distintas familias de funciones hash. Tomemos el ejemplo de SHA-1 y de la familia de funciones MD: para el primero, los bits del mensaje de inicio se utilizan con más frecuencia en la función de compresión, lo que hace que SHA-1 sea más seguro, pero un poco más lento. Puede encontrar un breve resumen de las funciones de hash en [Haitsma, 2001].

## **4. Un poco de historia**

En el siglo XVI, las capacidades matemáticas se demostraban en la plaza pública durante los desafíos. Estos duelos de científicos se organizaban en presencia de testigos, jueces, árbitros, etc. En estos desafíos, había gloria y dinero en juego. Por eso, todos los descubrimientos principales se guardaban celosamente en secreto. Cuando un matemático hacía un descubrimiento nuevo, se ponía en contacto con el comité de los desafíos públicos y afirmaba que era capaz de resolver una determinada clase de problemas. El comité organizaba un desafío donde los pretendientes se comprometían a proponer una solución.

# Criptografía y curvas elípticas

Hasta ahora hemos descrito el desarrollo de los distintos métodos de criptografía clásicos y modernos, basados en álgebra y aritmética. Es decir, estos métodos están vinculados con las propiedades de los números y su congruencia.

En esta sección, vamos a analizar algunas fronteras nuevas que se han abierto recientemente a la criptografía, en particular todo lo relacionado con los ámbitos de la seguridad y del criptoanálisis. Esto se debe a la interacción entre el álgebra clásica y las nociones de geometría, como el análisis de algunas curvas planas llamadas curvas elípticas.

## **1. Curvas algebraicas en un plano afín**

En lugar de considerar un campo finito , consideramos un campo arbitrario . Así podemos definir un plano afín , con coordenadas en este campo  (Sernesi, 1992). En efecto, podemos hacer la analogía con el caso donde , que induce un plano usual  con coordenadas cartesianas .

En el plano afín , practicamos la geometría de la misma manera que en un plano cartesiano real. Consideramos las curvas algebraicas que pertenecen a  y que están definidas por ecuaciones de la forma:



Donde  es un polinomio con coeficientes definidos en , lo que supone que no son constantes y no tienen factores recurrentes. La curva definida por la ecuación anterior es un conjunto de puntos  tales que . Claramente, si sustituimos  por  donde , obtendremos la misma curva. La curva definida por la ecuación (1) se denomina irreductible si el polinomio es irreductible en  (un polinomio que no es ni inversible, ni el producto de dos polinomios no inversibles). Este concepto es dependiente el campo , porque un polinomio puede ser irreductible en , pero no en una extensión de .

*Ejemplo:*

Una curva en  con la ecuación  es irreductible porque  es un polinomio en . Por el contrario, la misma ecuación en  es reductible porque  en .

Si  tiene un grado *d*, en este caso *d* es el grado de la curva de ecuación (1). Las curvas de grado 1 se denominan lineales, las de grado 2 cónicas, las de grado 3 cúbicas y así sucesivamente.

Líneas y curvas racionales

En el campo  se pueden identificar todas las líneas rectas. Vamos a tomar el ejemplo de la línea *R* con la siguiente ecuación.



donde  y  son distintas de 0. Considerando que , podemos proyectar la línea *R* en el eje *x*, asociando a cada punto  de *R* el punto  del eje *x*, que se identificará con . Este tipo de mapeado es biyectivo. En efecto, conociendo *u*, podemos volver a , si el punto  pertenece a *R*. La proyección viene dada por:



Y el "mapeado" inverso viene dado por:



De manera análoga, si , la línea *R* se puede proyectar en el eje *y*.

En conclusión, desde nuestro punto de vista, las líneas no son interesantes porque, en criptografía, buscamos encontrar grupos distintos de .

Dada la curva *C* con la siguiente ecuación:



donde  y  son polinomios en *x* distintos de 0 y no tienen factores en común. Este tipo de curva es irreductible. La proyección se define de la misma manera:



Y la proyección inversa viene dada por:



Hay que destacar que  no está definida cuando . Estrictamente hablando, no es la inversa de , porque es la inversa de una infinidad de puntos, salvo para . Este tipo de curva se denomina racional.

De nuevo, las curvas racionales son muy parecidas a  y no son interesantes en criptografía.

Curvas hiperelípticas

En este caso, las curvas vienen dadas por proyecciones en el eje x que no tienen inversas. Estas curvas se denominan curvas hiperlípticas. Un ejemplo de este tipo de curva viene dado por las curvas irreductibles de grado superior a 2 y tienen una ecuación de la siguiente manera:

 (2)

donde  y . Si  es un punto de la curva *C*, esto implica que *v* es la solución de la ecuación.



que generalmente tiene dos soluciones distintas.

Si  tiene características distintas de 2, entonces la ecuación anterior tiene una solución única si el discriminante es igual a 0, que es, si y solo si:  (3).

La ecuación (3) es válida para todo punto . Sin embargo, si consideramos un conjunto *C’* de puntos  de tal manera que existe un punto  bastante grande (más grande que el grado del polinomio), esto implica que  es el polinomio cero, y lleva a una contradicción porque, en este caso, encontramos:



con la hipótesis de que *C* es irreductible.

La conclusión es que las curvas hiperelípticas se pueden considerar como una doble cobertura de . Este concepto es especialmente verdad cuando  es algebraicamente cerrado. En este caso, si  tiene características distintas de 2, tenemos exactamente dos puntos de *C* a través del punto *u* en el eje *x*. Y si  es la raíz de , el punto único  de *C* corresponde a esta raíz.

En el caso donde  tiene características de 2, para todo  que no es ráiz del polinomio , entonces tenemos exactamente dos puntos de *C* a través del punto *u* en el eje *x*. Y si  es la raíz de  el punto único  de *C* corresponde a esta raíz.

Obviamente, debemos considerar curvas con un comportamiento de las proyecciones más complejo. Por el momento, la imagen recíproca de un punto de  puede tener un tamaño mayor de 2. Pero no tenemos que explorar estos casos, porque buscamos considerar curvas construibles de manera sencilla.

Curvas elípticas

Sean  y  dos funciones de forma sencilla y compatible con la hipótesis de que la curva es de un grado superior a 2. En efecto, si suponemos que  de grado 1 y que , la curva *C* de ecuación 2 es cúbica.

Es importante observar que con un cambio de variable adecuado, la ecuación de la curva *C* se puede simplificar:

**Proposiciones**: sea C una curva de ecuación:



Así es posible cambiar de coordenadas internamente, de tal manera que en el nuevo sistema de coordenadas:

* si  tiene características distintas de 2 o de 3, *C* tiene una ecuación con la forma:



* si  tiene características de 3, *C* tiene una ecuación con la forma:



* si  tiene características de 2, *C* tiene una ecuación con la forma:



o con otras formas:



Porque, es interesante observar que si  tiene características distintas de 2, podemos proceder al siguiente cambio de variables:



Entonces la ecuación de *C* toma la forma de la ecuación 4, donde . Y si las características son distintas de 3, procedemos al siguiente cambio de variables:



Actualmente la ecuación de *C* tiene la forma de la ecuación 5. Ahora suponemos que  tiene características de 2. Utilizando el cambio de variable anterior, entonces la ecuación de *C* toma la forma:



Si *m* = 0, la ecuación tiene la forma de la ecuación 7. Y si *m* ≠ 0, en este caso, procedemos al siguiente cambio de variables:



para obtener una ecuación con la forma de la ecuación 8.

Las ecuaciones de la forma 5 a la 8 se llaman ecuaciones canónicas y responden a la definición de Weierstrass de una curva cúbica. Estas ecuaciones denotan un tipo de curva *E* llamada curva elíptica (Silverman, 1985).

La figura 3.12 representa una ilustración de curva elíptica.

Diagrama, Dibujo de ingeniería

Descripción generada automáticamente

*Figura 3.12 - Curva elíptica de la ecuación .*

¿Por qué son interesantes estas curvas para la criptografía? Para responder a esta pregunta, ahora vamos a analizar la pertinencia y la aplicación de estas curvas elípticas en el mundo del cifrado.

## **2. Curvas elípticas**

Volviendo a la criptografía. Ahora, tenemos a nuestra disposición un grupo multiplicativo del campo , pero también curvas elípticas que se pueden definir en este grupo. Esta configuración nos permite tener una seguridad mejor de nuestro sistema criptográfico.

Pero se plantea una pregunta: ¿las curvas elípticas en campo finito son apropiadas para la criptografía? La respuesta es "principalmente favorable".

Las curvas elípticas vienen dadas de manera concreta, bastaría con tener sus ecuaciones. Pero eso no significa que la determinación de sus puntos sea una tarea fácil. En efecto, no hay algoritmos polinómicos para generar puntos de curvas elípticas en un campo . Sin embargo, hay algoritmos probabilísticos capaces de determinar puntos en una curva elíptica con una probabilidad elevada (Schoof, 1985). También tenemos que mencionar el hecho de que el algoritmo polinómico debido a (Schoof, 1985) permite calcular el número de puntos de una curva elíptica en , pero sin determinarlos.

La exponenciación en una curva elíptica *E*, que vuelve a multiplicar un punto de la curva *E* por un número entero, se puede hacer en tiempo polinómico.

Menezes, Okamoto y Vanstone (Menezes, 1993) demostraron que el problema del logaritmo discreto en una curva elíptica no es más fácil que en un campo finito. Y con mayor motivo, la hipótesis de Diffie-Hellman se verifica para una curva elíptica en un campo .

Para terminar, vamos a ver un ejemplo para mostrar cómo es posible intercambiar claves privadas utilizando un sistema RSA basado en una curva elíptica.

Elegimos un campo , una curva elíptica *E* y un punto , hechos públicos. Para el buen funcionamiento del sistema, es importante que *E* tenga muchos puntos en .

Cada usuario *U* elige una clave  y un número entero positivo, que guarda en privado. Publica , que es un punto de la curva *E*, calculado en tiempo polinómico. Conviene tener  si . Para ello, es importante tener  con un orden elevado, si es posible más elevado que el número de usuarios del sistema. Por lo tanto, cuando se elige  de manera aleatoria, entonces los puntos  serán distintos.

Si dos usuarios *A* y *B* desean intercambiar una clave privada, pueden hacerlo poniéndose de acuerdo sobre el uso, como clave privada, de un punto . En este caso, los dos pueden determinar esta clave en tiempo polinómico. Teniendo en cuenta la dificultad de calcular un logaritmo discreto en *E* y la hipótesis de Diffie-Hellman, el cálculo de  no puede realizarlo otra persona.

Es obvio que *A* y *B* pueden utilizar como clave privada el número deducido de , una de las coordenadas de este punto, su suma o cualquier otra aplicación sobre estas coordenadas. Este método se aplica muy bien al sistema RSA.

Para resumir los resultados de la aplicación de las curvas elípticas en criptografía, podemos afirmar que una de las principales aplicaciones es el intercambio de claves privadas de manera segura para utilizar un sistema de cifrado como RSA.

Para terminar, las curvas elípticas también encuentran una aplicación en las pruebas de primalidad o de factorización (Koblitz, 1994).

## **3. Criptografía y teoría del caos**

Terminamos este capítulo sobre el encriptado con una introducción a la criptografía caótica, un campo donde se están realizando intensas investigaciones.

La criptografía caótica es una alternativa a la criptografía algorítmica. Se basa en una dinámica muy sensible a las condiciones iniciales y su comportamiento puede parecer aleatorio. Estas dos características son adecuadas para cifrar información.

El caos es una noción matemática oscura propuesta en los años 60. A menudo se admite erróneamente que el caos es un proceso aleatorio o estocástico. También hay que saber que un proceso donde el azar no tiene cabida y puede dar un resultado que parece perfectamente aleatorio, se llama caos.

El caos pertenece a una amplia categoría de procesos dinámicos llamada sistema determinista. Un sistema dinámico se puede describir mediante una posición inicial y una ley de evolución que da todas las posiciones posibles de un punto en desplazamiento.

El caos presenta dos características: una ínfima variación de la posición inicial induce rápidamente una diferencia muy grande en las trayectorias y toda posición inicial es infinitamente cercana a una trayectoria periódica. Resumiendo, se denomina caótica a todo sistema dinámico determinista no lineal y que presenta las dos características anteriores: una gran sensibilidad a las condiciones iniciales y la presencia de órbitas periódicas.

La idea de utilizar el caos en criptografía y en el ámbito de la protección de la información está vinculada a su carácter pseudoaleatorio. La naturaleza imprevisible, estadística o entrópica del caos lo convierte en un candidato excelente para interactuar con la teoría de la información, pero también con los ámbitos del tratamiento de la señal donde se utilizan nociones de escalonamiento espectral. En este último caso, se utilizan mecanismos de sincronización y de desincronización para encriptar y desencriptar las señales. En la criptografía de señales digitales, se utilizan series logísticas definidas por relaciones de recurrencia para generar de manera caótica matrices que entran dentro de la construcción de funciones de hash (Wang, 2008).