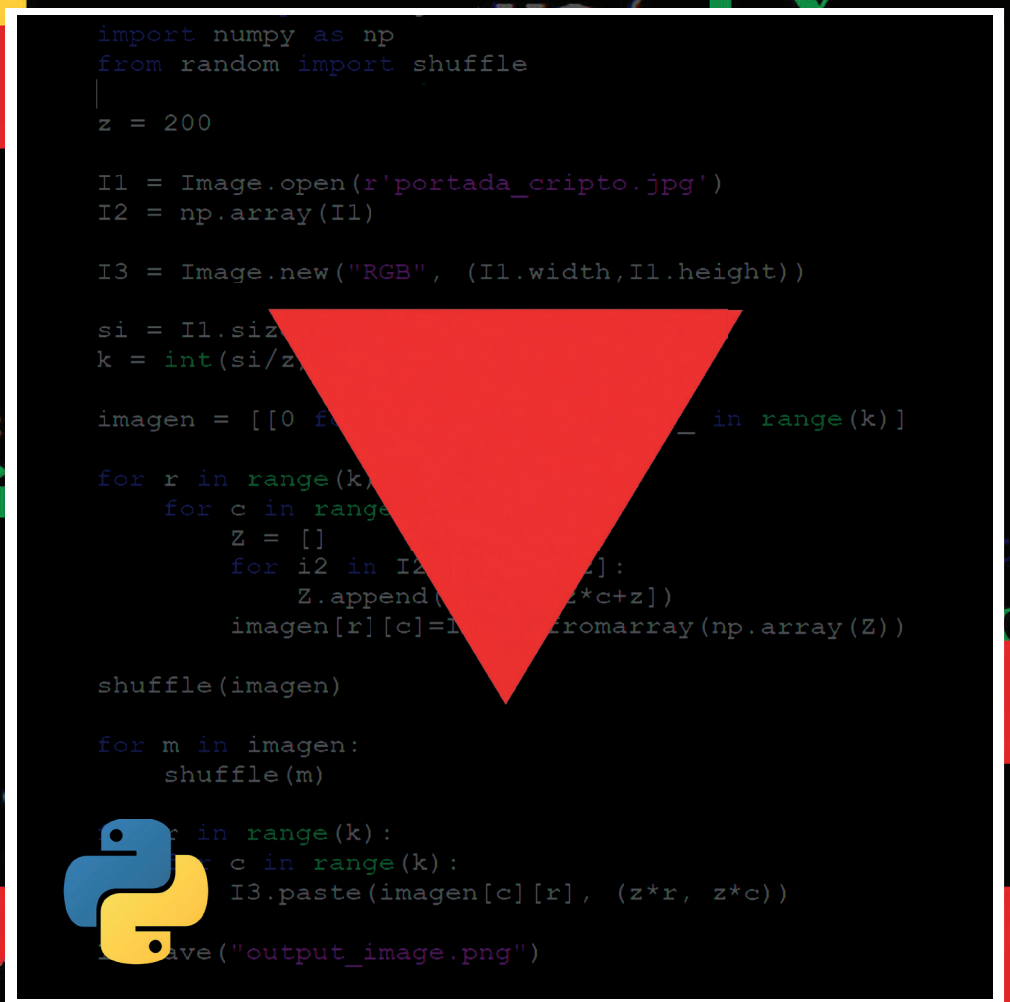
Introducción a la criptografía

Explicaciones, algoritmos y códigos en *Python*



Iván O. Sosa Pérez
Raúl A. Espejel Morales

Introducción a la criptografía

Explicaciones, algoritmos y códigos en *Python*

Introducción a la criptografía

Explicaciones, algoritmos y códigos en *Python*

Iván O. Sosa Pérez
Raúl A. Espejel Morales

UACM
Universidad Autónoma
de la Ciudad de México
NADA HUMANO ME ES AJENO

Biblioteca
del
Estudiante

Ficha catalográfica E-S/N

Sosa Pérez, Iván O., autor

Introducción a la criptografía : explicaciones, algoritmos y códigos en Python / Iván O. Sosa Pérez, Raúl A. Espejel Morales. Primera edición. México, D.F. : Universidad Autónoma de la Ciudad de México, 2025.
119 páginas : ilustraciones ; 23 cm.
Bibliografía : páginas 117-119.

ISBN: 978-607-2615-78-6

1. Encriptamiento de datos (Computación). 2. Seguridad informática. 3. Firmas digitales. 4. Comunicaciones digitales. 5. Python (Lenguaje de programación para computadora). I. Espejel Morales, Raúl A., autor. II. Título.

LC QA268 Dewey 003.54

Introducción a la criptografía explicaciones, algoritmos y códigos en Python

primera edición, 2025

García Diego 168, col. Doctores,
alc. Cuauhtémoc, c. p. 06720, México, D F

ISBN: 978-607-2615-78-6

https://www.uacm.edu.mx/Organizacion/CoordinacionAcademica/Biblioteca_Estudiante

Material educativo universitario de distribución gratuita para estudiantes de la UACM. Prohibida su venta

Hecho e impreso en México

Índice

Presentación

La criptografía tiene una larga y fascinante historia. La utilización de técnicas criptográficas abarca más de cuatro mil años de la historia humana y ha sido empleada tradicionalmente por los militares, los bancos, los comerciantes, los servicios diplomáticos, y en general por los gobiernos desde tiempos antiguos para asegurar sus comunicaciones y así proteger sus secretos. La criptografía está conformada por un conjunto muy amplio de técnicas que sirven para la protección de la información y emplean diversos conceptos y técnicas matemáticas relacionadas con la confidencialidad, la integridad y la autenticidad de un mensaje.

Actualmente, debido al uso extendido de la tecnología informática, resulta fundamental la protección de la información digital. La protección de la información se realiza mediante diversas técnicas de criptografía o encriptación, por lo que es importante el estudio de éstas. Existen diferentes productos de seguridad desarrollados para satisfacer las necesidades de una sociedad con un empleo intensivo de información digital. Sin embargo, los conocimientos fundamentales para comprender y emplear de manera adecuada dicha tecnología son poco difundidos y existen muy pocos libros en español que nos permitan acercarnos a los conceptos, las técnicas y los algoritmos empleados en la criptografía en general y en la criptografía práctica de manera específica.

El propósito del presente libro es proporcionar una introducción a la criptografía y al criptoanálisis. El libro trata sobre cómo establecer comunicaciones seguras entre dos o más partes, de tal manera que exista un nivel razonable de confidencialidad, integridad y autenticidad en los datos intercambiados. El proyecto de elaboración de un libro sobre el tema surge de la necesidad de integrar una serie de apuntes realizados al impartir diversos talleres sobre criptografía y comunicaciones seguras que implican intercambio de información en los servicios de comunicación digitales (Telegram,

WhatsApp, Signal, etc), así como la seguridad de los datos almacenados en las nubes, y en general, el flujo de información en la web. Los autores consideran que, a pesar de que la bibliografía sobre el tema es amplia y existen varios textos de introducción, la gran mayoría están escritos en inglés. Frente a la necesidad de contar con un texto de trabajo en español, los autores decidieron darle cuerpo a los apuntes elaborados en el trabajo de preparación del tema.

Se brinda un documento en donde, a través de cada capítulo, se exponen las relaciones entre diferentes aspectos de la criptografía considerando el contexto, el sustento matemático, el algoritmo y la perspectiva de cada una de las técnicas criptográficas. De tal manera que se busca presentar los algoritmos implementados en código *Python 3*, como un mecanismo para integrar conocimientos; estos programas pueden descargarse del sitio *GitHub*, cuya liga se encuentra al final del Apéndice A. Este libro contiene un apéndice elaborado con el propósito de brindar las herramientas necesarias para comenzar a programar y aprender de los códigos que se presentan, ésto representa una ventaja importante ya que, en la actualidad, programar en un lenguaje de alto nivel es una habilidad indispensable para los estudiantes de ciencia y tecnología. Las ideas conceptuales vertidas en este texto, así como la descripción de su contexto, se exponen con la intención de dirigirse hacia su concretización, buscando aportar a los lectores una formación básica y aplicada en criptografía.

Consideramos que este libro apoya directamente el aprendizaje autónomo de los estudiantes, mediante la integración interdisciplinaria de conocimientos asociados a cursos sobre ciencias básicas e ingenierías, así como ciencias de la computación, por ejemplo, modelación matemática programación, teoría de la información, entre otras.

Capítulo 1

Definiciones y conceptos

En este capitulo definiremos algunos términos asociados a conceptos básicos como *emisor* y *receptor*, *mensaje* y *encripción*. Después hablaremos de *llaves*, *algoritmos criptográficos* y su seguridad, para después explicar en qué consiste el *criptoanálisis*.

El problema fundamental en criptografía es el siguiente: supongamos que un emisor necesita enviar un mensaje a cierto receptor, y éste debe ser enviado de manera segura. El contenido del mensaje en sí debe permanecer seguro, esto es, de alguna forma, el emisor necesita estar seguro de que sólo el receptor, a quién se le envió el mensaje, pueda leerlo [1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 16].

1.1. Terminología básica

Para resolver este problema, es frecuente trabajar con un conjunto de símbolos al que llamaremos alfabeto y lo denotaremos por **A**. Por ejemplo, este conjunto puede estar formado por las letras del alfabeto en español.

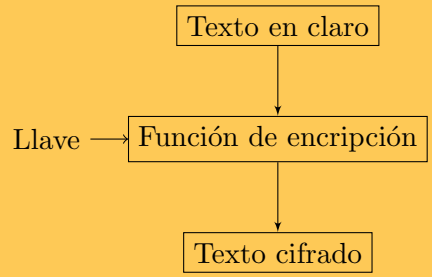
A las *palabras* o al conjunto de cadenas de símbolos de un alfabeto le llamamos *espacio de mensajes* y lo denotamos por **M**. A algún subconjunto de **M** que tenga significado para el *emisor*, le llamaremos *texto en claro*. Al proceso de "disfrazar" un texto en claro, transformándolo en otras cadenas que estarán contenidas en un conjunto **C**, se le llama encripción o encriptación. El conjunto **C** es también un espacio de mensajes, cuyo alfabeto puede o no diferir de **A**. Dicho mensaje encriptado es llamado también *mensaje cifrado* o *criptograma*, y al proceso de recuperar el texto cifrado y conver-

tirlo de nuevo en el texto en claro se le llama *desencripción*. Estos procesos también son llamados *cifrado* y *descifrado* de un mensaje.

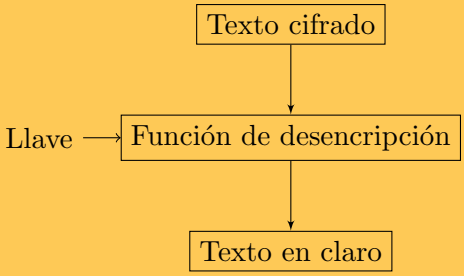
El texto sin cifrar puede ser, por ejemplo, una secuencia de bits, un archivo de texto, un archivo con voz digitalizada, una imagen o video digital, etcétera.

Formalmente, podemos definir una función $E_e : \mathbf{M} \to \mathbf{C}$ que es llamada *función de encripción* o *transformación de encripción* que contiene un elemento e, llamado **llave**, indispensable para llevar a cabo esta transformación. Al conjunto de estos elementos se le llama *espacio de llaves* y lo denotaremos por $\mathbf{K}$. A la función $D_d : \mathbf{C} \to \mathbf{M}$ que recupera un texto en claro a partir de uno cifrado, mediante la llave d, le llamamos de desencripción. Definimos como **criptosistema** a la quíntupla $(\mathbf{M}, \mathbf{C}, \mathbf{K}, \mathbf{E}_e, \mathbf{D}_d)$. Las definiciones anteriores se representan en el siguiente diagrama:

Cuadro 1.



Cuadro 2.



Lo anterior se reduce a que existe un proceso o función de encripción E_e, que opera sobre M para producir C, esto es:

$$E_e(M) = C.$$

En el proceso inverso, la función de desencripción D_d opera sobre el mensaje cifrado C para producir el texto llano M:

$$D_d(C) = M.$$

Ya que todo el proceso de cifrar y descifrar un mensaje busca finalmente poder recuperar el texto en claro, la siguiente identidad debe satisfacerse:

$$D_d(E_e(M)) = M.$$

A la ciencia y arte de mantener un mensaje seguro se le llama **criptografía**, por otro lado, el **criptoanálisis**, se encarga de romper o debilitar la seguridad de los textos cifrados con el objeto de descifrarlos. La unión de las matemáticas, la teoría de la computación y el cómputo aplicado, que abarca a la criptografía y el criptoanálisis, es conocida como **criptología**.

1.2. Autenticación, integridad y responsabilidad de propiedad

Además de proporcionar confidencialidad a un mensaje, la criptografía debe asegurarse de proporcionar otras características que son vitales para la comunicación segura, sean éstas presenciales o a través de computadoras. Por ejemplo, el poder verificar la identidad de alguien, esto es, que sus credenciales o mecanismos de identidad sean válidos, o que los documentos que alguien envía, efectivamente provengan de esa persona, estén completos y correspondan al documento original. Estas características se pueden describir como sigue:

Autenticidad: debe ser posible para el receptor de un mensaje poder comprobar su origen, de tal manera que un intruso no pueda suplantar a alguien más.

Integridad: debe ser posible para el receptor del mensaje verificar que éste no ha sido modificado durante la transmisión. Un intruso no debe ser capaz de sustituir un mensaje por otro.

Responsabilidad de propiedad: quien envía un mensaje no debe ser capaz de negar falsamente que lo envió.

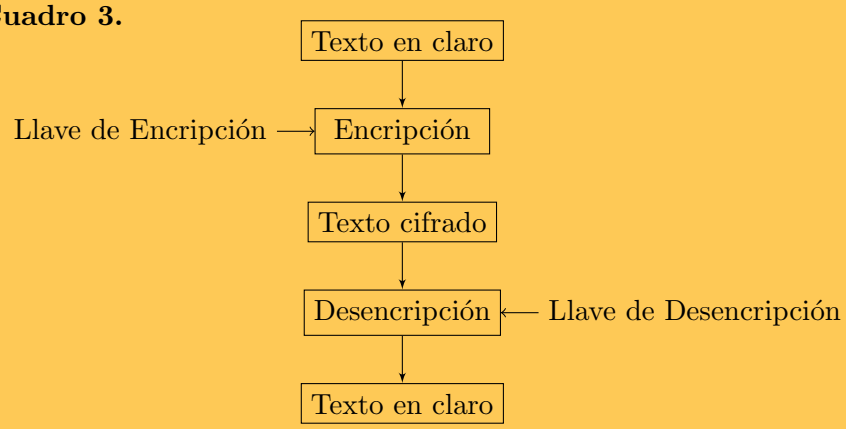
Ademas un sistema criptografico debe:

- Ser eficiente y fiable.
- Transmitir, almacenar y transferir archivos por algún canal de datos.
- Mantener la llave en secreto aún cuando el algoritmo de cifrado sea público. Su seguridad reside exclusivamente en la secrecía de la llave.
- Basar su fortaleza en la imposibilidad de romper el cifrado o de encontrar la llave a partir de otros datos de carácter público.

Si la seguridad de un algoritmo está basada únicamente en mantener en secreto su funcionamiento, le llamaremos **restringido**. Este tipo de algoritmos tienen actualmente sólo interés histórico, ya que son inadecuados para los estándares de seguridad actuales.

La criptografía moderna basa la seguridad en el empleo de una o más llaves y no en el conocimiento de los detalles del algoritmo empleado para cifrar. Esto permite que éste sea publicado, analizado y sometido a pruebas por un conjunto amplio de personas interesadas en él sin vulnerar su seguridad. Si alguien externo al grupo que emplea un algoritmo de este tipo quisiera leer uno de los mensajes no podrá hacerlo, aun conociendo los detalles del algoritmo, a menos que conozca la llave. La seguridad en la comunicación de los mensajes está basada en la solidez del algoritmo y en la secrecía de las llaves. Algunos algoritmos emplean una llave diferente para la encripción y la desencripción, ver diagrama en el cuadro 3:

Cuadro 3.



1.3. Algoritmos criptográficos simétricos

En este tipo de algoritmos, también llamados convencionales o **de llave secreta**, las llaves para cifrar y descifrar son iguales, o bien, una puede ser calculada a partir de la otra. En ellos es fundamental que el emisor y el receptor del mensaje acuerden la llave antes de que se puedan comunicar de forma segura. La seguridad de un algoritmo simétrico descansa en la secrecía de la llave, divulgar la llave significa que cualquiera, con conocimiento del algoritmo, puede encriptar y desencriptar mensajes.

Los algoritmos simétricos pueden dividirse en dos categorías: Los que operan sobre una sola letra del texto llano a la vez, por lo que son llamados **secuenciales**, y los que operan sobre bloques o grupos de letras del texto en claro y que son llamados **de bloque**. Antes del desarrollo de las computadoras, los algoritmos generalmente operaban sobre el texto llano codificando una letra o caracter por vez, es decir como un algoritmo secuencial.

Ejemplo 1. Un criptograma empleando el Cifrado del César

Un algoritmo de cifrado consiste en cambiar cada una de las letras de un mensaje por otra, la cual se encuentra un cierto número e de posiciones adelante en el orden alfabético,

$$\mathbf{A} = \{A, B, C, \ldots, X, Y, Z\}.$$

Definimos el espacio de mensajes $\mathbf{M}$ como las cadenas de letras elemen-

tos de **A**. De esta forma, agrupando el mensaje "LA GUERRA ZOMBI TERMINÓ" en una cadena, tenemos:

$$m = \text{LAGUERRAZOMBITERMINO}$$

La llave de este algoritmo es un número entero y en este caso $e = 3$, entonces, el mensaje cifrado es:

$$c = E_e(m) = \text{"ODJXHUUDCRPELWHUPLQR"}.$$

Notemos que la letra Z se transforma en C, puesto que las últimas letras se relacionan con las primeras, como si el alfabeto estuviera escrito sobre una banda. Más adelante se introducirá la función *módulo* para implementar éste y otros algoritmos.

1.4. Algoritmos de llave pública

Este tipo de algoritmos están diseñados para que la llave de encripción sea diferente de la llave de desencripción, por esta razón son llamados **asimétricos**. Están hechos de forma tal que la llave de desencripción no pueda ser calculada u obtenida a partir de la llave de encripción. Son llamados de **llave pública** porque la llave de encripción puede hacerse pública sin comprometer la seguridad del criptosistema; cualquier persona puede emplear esta llave para codificar un mensaje, pero sólo la persona que posea la llave de desencripción puede decodificarlo. En este tipo de sistemas la llave de encripción suele llamarse *llave pública*, mientras que la llave de desencripción es llamada *llave privada*. En algunas situaciones es posible encriptar un mensaje empleando la llave privada y descifrarlo usando la llave pública, esto es, se usa la llave privada como un cifrado y una *firma digital*.

Una analogía que se emplea para explicar la idea fundamental de la criptografía de llave pública se debe a Simon Singh y es conocida como *el problema de la mezcla de colores* [3]:

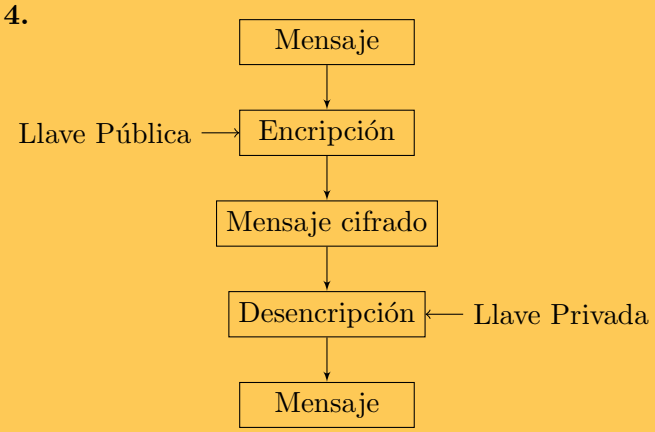
Cita 1. El problema de la mezcla de colores, Simon Singh.

Supongamos que todo el mundo tiene un bote de tres litros que contiene un litro de pintura amarilla. Si dos personas quieren acordar una clave secreta, cada uno añade un litro de un color secreto propio a su bote. Entonces cada uno envía al otro su bote con la mezcla. Finalmente, le añaden al bote del otro su propio color secreto. Ahora ambos botes

tienen la misma mezcla de color, con la particularidad de que ninguno tiene idea del cuál es el color secreto del otro. Además, si una tercer persona intercepta el bote en cualquiera de sus comunicaciones, no puede deducir su color porque desconoce los colores secretos.

Más adelante se abordará el sistema criptográfico RSA y los protocolos de intercambio de llaves que utilizan algoritmos asimétricos, por lo que estos conceptos se desarrollarán de manera detallada en dichas secciones. A continuación presentamos un esquema del uso de estas llaves. Este mecanismo es más usado para encriptar las llaves de sesión que los mensajes en sí, los cuales se suelen encriptar empleando un sistema de llave simétrica, o formando un sistema híbrido.

Cuadro 4.



1.5. Criptoanálisis

El tema central de la criptografía es mantener el texto claro y la llave a salvo de atacantes que puedan estar interesados en obtenerlos. Supondremos que dichos atacantes tienen acceso completo a las comunicaciones entre el receptor y el emisor. El criptoanálisis tiene como propósito recuperar el texto llano de un mensaje cifrado sin tener acceso a la llave. El criptoanálisis es exitoso si se recupera el texto claro o la llave, para lo cual debe encontrar las debilidades del criptosistema que se esté atacando. Al intento de realizar un criptoanálisis se le llama ataque criptoanalítico, o simplemente, ataque.

Criptoanálisis de un mensaje secreto

A continuación se incluye un fragmento del texto "El escarabajo de oro", de Edgar Allan Poe [4], en el cual se realiza el criptoanálisis de un mensaje secreto.[1]

Cita 2. Criptoanálisis en "El Escarabajo de oro" de Edgar Allan Poe.

... Y al llegar aquí, Legrand, habiendo calentado de nuevo el pergamino, lo sometió a mi examen. Los caracteres siguientes aparecían de manera toscamente trazada, en color rojo, entre la calavera y la cabra:

53‡‡†305))6*;4826)4‡○)4‡);806*;48†8π60))85;1‡(;:*8†83
(88)5*†;46(;88*96*?;8)*‡(;485);5*†2:*‡(;4956*2)5*–4)8
π8*;4069285);)6†8)4‡‡;1(‡9;48081;8:8‡1;48†85;4)485†52
8806*81(‡9;48;(88;4(‡?34;48)4‡;161;:188;‡?;

—Pero —dije, devolviéndole la tira—sigo estando tan a oscuras como antes. Si todas las joyas de Golconda esperasen de mí la solución de este enigma, estoy en absoluto seguro de que sería incapaz de obtenerlas.
—Y el caso—dijo Legrand—que la solución no resulta tan difícil como cabe imaginarla tras del primer examen apresurado de los caracteres. Estos caracteres, según pueden todos adivinarlo fácilmente forman una cifra, es decir, contienen un significado pero por lo que sabemos de Kidd, no podía suponerle capaz de construir una de las más abstrusas criptografías. Pensé, pues, lo primero, que ésta era de una clase sencilla, aunque tal, sin embargo, que pareciese absolutamente indescifrable para la tosca inteligencia del marinero, sin la clave.

—¿Y la resolvió usted, en verdad?

—Fácilmente; había yo resuelto otras diez mil veces más complicadas. Las circunstancias y cierta predisposición mental me han llevado a interesarme por tales acertijos, y es, en realidad, dudoso que el genio humano pueda crear un enigma de ese género que el mismo ingenio humano no resuelva con una aplicación adecuada. En efecto, una vez que logré descubrir una serie de caracteres visibles, no me preocupó apenas la simple dificultad de desarrollar su significación.

[1] Se presenta el texto íntegro ya que, en las diferentes traducciones al español, aparecen errores en la tipografía que conducen a errores en el criptoanálisis. Es la primera vez que se presenta el texto de manera adecuada, de modo que el análisis criptográfico es consistente.

En el presente caso —y realmente en todos los casos de escritura secreta—la primera cuestión se refiere al lenguaje de la cifra, pues los principios de solución, en particular tratándose de las cifras más. sencillas, dependen del genio peculiar de cada idioma y pueden ser modificadas por éste. En general, no hay otro medio para conseguir la solución que ensayar (guiándose por las probabilidades) todas las lenguas que os sean conocidas, hasta encontrar la verdadera. Pero en la cifra de este caso toda dificultad quedaba resuelta por la firma. El retruécano sobre la palabra Kidd sólo es posible en lengua inglesa. Sin esa circunstancia hubiese yo comenzado mis ensayos por el español y el francés, por ser las lenguas en las cuales un pirata de mares españoles hubiera debido, con más naturalidad, escribir un secreto de ese género. Tal como se presentaba, presumí que el criptograma era inglés.

Fíjese usted en que no hay espacios entre las palabras. Si los hubiese habido, la tarea habría sido fácil en comparación. En tal caso hubiera yo comenzado por hacer una colación y un análisis de las palabras cortas, y de haber encontrado, como es muy probable, una palabra de una sola letra (a o I-uno, yo, por ejemplo), habría estimado la solución asegurada. Pero como no había espacios allí, mi primera medida era averiguar las letras predominantes así como las que se encontraban con menor frecuencia. Las conté todas y formé la siguiente tabla:

Símbolo	Frecuencia
8	33
;	26
4	19
)	15
‡	14
*	13
5	12
6	11
(	10
†	8
1	8
0	6
9	5
2	5
:	4
3	4
?	3
π	2

Ahora bien: la letra que se encuentra con mayor frecuencia en inglés es la *e*. Después, la serie es la siguiente:

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

La **e** predomina de un modo tan notable, que es raro encontrar una frase sola de cierta longitud de la que no sea el carácter principal.
Tenemos, pues, nada más comenzar, una base para algo más que una simple conjetura. El uso general que puede hacerse de esa tabla es obvio, pero para esta cifra particular sólo nos serviremos de ella muy parcialmente. Puesto que nuestro signo predominante es el **8**, empezaremos por ajustarlo a la **e** del alfabeto natural. Para comprobar esta suposición, observemos si el **8** aparece a menudo por pares —pues la **e** se dobla con gran frecuencia en inglés —en palabras como, por ejemplo, **meet**, **speed**, **seen**, **been**, **agree**, etcétera. En el caso presente, vemos que está doblado lo menos cinco veces, aunque el criptograma sea breve. Tomemos, pues, el **8** como **e**. Ahora, de todas las palabras de la lengua, **the** es la más usual; por tanto, debemos ver si no está repetida la com-

binación de tres signos, siendo el último de ellos el **8**. Si descubrimos repeticiones de tal letra, así dispuestas, representarán, muy probablemente, la palabra **the**. Una vez comprobado esto, encontraremos no menos de siete de tales combinaciones, siendo los signos **;48**. Podemos, pues, suponer que **;** representa **t**, **4** representa **h**, y **8** representa **e**. Hemos dado ya un gran paso.

Acabamos de establecer una sola palabra; pero ello nos permite establecer también un punto más importante; es decir, varios comienzos y terminaciones de otras palabras. Veamos, por ejemplo, el penúltimo caso en que aparece la combinación **;48** casi al final del criptograma. Sabemos que el **;** que viene inmediatamente después es el comienzo de una palabra, y de los seis signos que siguen a ese **the**, conocemos, por lo menos, cinco. Sustituyamos, pues, esos signos por las letras que representan, dejando un espacio para el signo desconocido:

t□eeth

Debemos, lo primero, desechar el **th** como no formando parte de la palabra que comienza por la primera **t**, pues vemos, ensayando el alfabeto entero para adaptar una letra al hueco, que es imposible formar una palabra de la que ese **th** pueda formar parte. Reduzcamos, pues, los signos a

t□ee

y volviendo al alfabeto, si es necesario como antes, llegamos a la palabra **tree** (árbol), como la única que puede leerse. Ganamos así otra letra, la **r**, representada por **(**, más las palabras yuxtapuestas **the tree** (el árbol). Un poco más lejos de estas palabras, a poca distancia, vemos de nuevo la combinación **;48** y la empleamos como terminación de lo que precede inmediatamente. Tenemos así esta distribución:

the tree ;4(‡?34 the

o sustituyendo con letras naturales los signos que conocemos, leeremos esto:

the tree thr‡?3h the

ahora, si sustituimos los signos desconocidos por espacios blancos o por puntos, leeremos:

the tree thr□□□h the

y, por tanto, la palabra **through** (a través) resulta evidente por sí misma.

Pero este descubrimiento nos da tres nuevas letras, **o**, **u**, y **g**, representadas por **‡**, **?** y **3**.

Buscando ahora cuidadosamente en el criptograma combinaciones de signos conocidos, encontraremos no lejos del comienzo esta disposición:

†83(88, o bien, **†egree**

que es, evidentemente, la terminación de la palabra **degree** (grado), que nos da otra letra, la **d**, representada por **†**.

Cuatro letras más lejos de la palabra **degree**, observamos la combinación,

;46(;88

cuyos signos conocidos traducimos, representando el desconocido por espacios, como antes; y leemos:

th□rtee□

arreglo que nos sugiere acto seguido la palabra **thirteen** (trece) y que nos vuelve a proporcionar dos letras nuevas, la **i** y la **n**, representadas por **6** y *****.

Volviendo ahora al principio del criptograma, encontramos la combinación.

53‡‡†

Traduciendo como antes, obtendremos

□good

Lo cual nos asegura que la primera letra es una **A**, y que las dos primeras palabras son **A good** (un bueno, una buena).

Sería tiempo ya de disponer nuestra clave, conforme a lo descubierto, en forma de tabla, para evitar confusiones. Nos dará lo siguiente:

Símbolo	Significado
5	a
†	d
8	e
3	g
4	h
6	i
*	n
‡	o
(	r
;	t

Tenemos así no menos de diez de las letras más importantes representadas, y con esto es suficiente, no es necesario proseguir con los detalles para encontrar la solución. Ya le he dicho lo suficiente para convencerle de que los criptogramas de ese género son de fácil solución, y para darle algún conocimiento de su desarrollo razonado. Pero tenga la seguridad de que la muestra que tenemos delante pertenece al tipo más sencillo de criptograma.

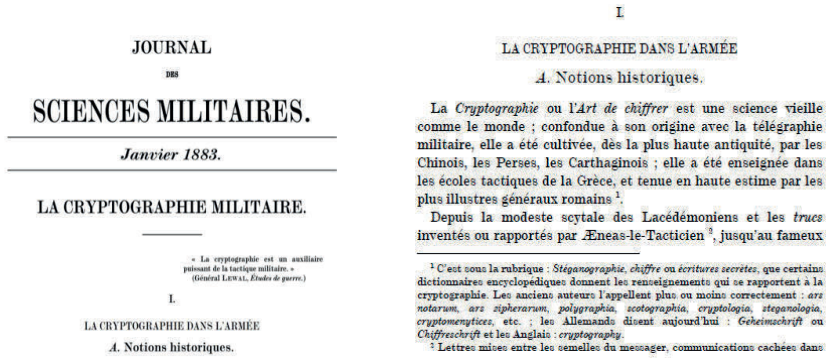
Sólo me queda darle la traducción entera de los signos escritos sobre el pergamino, ya descifrados. Hela aquí:

A good glass in the Bishop's Hostel in the devil's seat forty-one degrees and thirteen minutes northeast and by north main branch seventh, limb east side shoot from the left eye of the death's head a bee-line from the tree through the shot fifty feet out.

Edgar Allan Poe, *Narraciones Extraordinarias*, Porrúa, 2019.

La suposición fundamental del criptoanálisis fue formulada y propuesta en el siglo XIX por el alemán A. Kerckhoffs [5], y asume que el criptoanlista dispone de todos los detalles respecto al algoritmo criptográfico y su implementación, por lo que la seguridad de los mensajes reside completamente en mantener en secreto la llave. En su trabajo, un criptoanalista no siempre

dispone de información tan detallada. En la figura 1.1 se muestra la carátula y el resumen del artículo original.



JOURNAL

DES

SCIENCES MILITAIRES.

Janvier 1883.

LA CRYPTOGRAPHIE MILITAIRE.

« La cryptographie est un auxiliaire puissant de la tactique militaire. »
(Général LEWAL, *Études de guerre.*)

I.

LA CRYPTOGRAPHIE DANS L'ARMÉE

A. Notions historiques.

I.

LA CRYPTOGRAPHIE DANS L'ARMÉE

A. Notions historiques.

La *Cryptographie* ou l'*Art de chiffrer* est une science vieille comme le monde ; confondue à son origine avec la télégraphie militaire, elle a été cultivée, dès la plus haute antiquité, par les Chinois, les Perses, les Carthaginois ; elle a été enseignée dans les écoles tactiques de la Grèce, et tenue en haute estime par les plus illustres généraux romains [1].

Depuis la modeste scytale des Lacédémoniens et les *trucs* inventés ou rapportés par Æneas-le-Tacticien [2], jusqu'au fameux

[1] C'est sous la rubrique : *Stéganographie, chiffre* ou *écritures secrètes*, que certains dictionnaires encyclopédiques donnent les renseignements qui se rapportent à la cryptographie. Les anciens auteurs l'appellent plus ou moins correctement : *ars notarum, ars zipherarum, polygraphia, scotographia, cryptologia, steganologia, cryptomenytices*, etc. ; les Allemands disent aujourd'hui : *Geheimschrift* ou *Chiffreschrift* et les Anglais : *cryptography*.

[2] Lettres mises entre les semelles du messager, communications cachées dans

Figura 1.1: Carátula y resumen del artículo original de Kerckhoffs. Se reproduce por su valor histórico.

En este texto se presentan dos tipos de ataque. En ambos se supone que el criptoanalista tiene un conocimiento detallado del algoritmo de encripción:

- **Ataque de texto cifrado.** Este tipo de ataque es posible si se dispone de varios mensajes encriptados empleando el mismo algoritmo. El trabajo del criptoanalista consiste en desencriptar tantos mensajes como sea posible, o mejor aun, deducir las llaves de encripción empleadas.

  Este ataque se puede formular de la siguiente manera:

  Dado el conjunto de mensajes cifrados $C_1 = E_k(M_1), C_2 = E_k(M_2), \ldots C_i = E_k(M_i)$

  El propósito del ataque es deducir cualquiera de los mensajes $M_1, M_2, \ldots M_i$; las llaves k; o algún algoritmo para inferir el contenido de siguiente mensaje M_{i+1} a partir del texto cifrado $C_{i+1} = E_k(M_{i+1})$.

  Cuando se dispone no sólo del texto cifrado de varios mensajes, sino también de los mensajes en claro que los originaron, el trabajo consiste en deducir las llaves empleadas para encriptar los mensajes o en formular un algoritmo para desencriptar los mensajes empleando dichas llaves. La formulación de este tipo de ataque es la siguiente:

  Dado un conjunto de mensajes y sus respectivos mensajes cifrados $M_1, C_1 = E_k(M_1), M_2, C_2 = E_k(M_2), \ldots M_i, C_i = E_k(M_i)$.

  El propósito del ataque es deducir la o las llaves k o un algoritmo para inferir el mensaje M_{i+1} a partir de un nuevo texto cifrado

$C_{i+1} = E_k(M_{i+1})$.

- **Ataque sobre un texto llano elegido.** En este caso se dispone del texto cifrado de varios mensajes y de los textos en claro que los originaron, pero es posible elegir una parte del texto claro a analizar. Es decir, el criptoanalista puede escoger bloques o partes específicas del texto en claro para cifrarlos intentando emular el cifrado con el propósito de obtener la llave, se podrían entonces escoger aquellas partes que aporten mas información para este fin.

  La formulación es la siguiente:

  Dado un conjunto de mensajes y sus respectivos mensajes cifrados $M_1, C_1 = E_k(M_1), M_2, C_2 = E_k(M_2), \ldots M_i, C_i = E_k(M_i)$. En donde el criptoanalista puede elegir $M_1, M_2, ...M_i$ o fragmentos de éstos con el propósito de deducir las llaves.

Los ataques sobre un texto claro son bastante frecuentes ya que no es inusual obtener un mensaje de texto claro que ha sido encriptado, sobornando o forzando de alguna manera a alguien para que cifre un mensaje determinado y lo envíe de tal forma que sea posible recuperar el texto cifrado. Por ejemplo, en el ambiente diplomático, los canales de comunicación con frecuencia están encriptados; los diplomáticos emplean la encriptación para comunicar a sus países cierto tipo de información de tal forma que no es improbable hacerles llegar cierta información del interés de su país y monitorear el tráfico de mensajes. Históricamente estas técnicas ha sido empleadas, en particular se emplearon contra alemanes y japoneses durante la Segunda Guerra Mundial y durante la Guerra Fría.

También diversos mensajes tienen principios y finales estandarizados que podría ser conocidos por el criptoanalista. El código fuente encriptado es especialmente vulnerable debido al empleo regular de ciertas palabras como *for*, *next* , *else*, *return*, etc. El código ejecutable cifrado tiene vulnerabilidades semejantes ya que aparecen regularmente funciones, estructuras recurrentes, estructuras de bifurcación, etcétera.

En general, los algoritmos más seguros son aquellos que son públicos y que han sido atacados por los mejores criptógrafos del mundo durante años, han resistido sus ataques y han permanecido seguros. Es frecuente que los buenos equipos de criptógrafos hagan públicos los detalles de los algoritmos que desarrollan para ser sometidos a pruebas o ataques por la comunidad con el fin de establecer en qué medida sus algoritmos son seguros.

Un algoritmo es una secuencia finita y ordenada de instrucciones elementales que, dados los valores de entrada de un problema, en algún momento

finaliza y devuelve una solución. Un algoritmo criptográfico pretende complicar la tarea de un posible atacante haciendo que el número de operaciones necesarias para romper el cifrado sea tan grande que resulte impracticable. Si el costo para romper un algoritmo es mayor que el valor de los datos encriptados, entonces estos probablemente están seguros. Si el tiempo necesario para romper un algoritmo es mayor que el tiempo en el cual los datos cifrados deben permanecer secretos o si la cantidad de datos encriptados con una sola llave es menor que la cantidad de datos necesarios para romper el algoritmo, entonces es probable que los datos estén seguros.

Una de las clasificaciones frecuentemente empleadas para establecer las categorías asociadas a romper un algoritmo criptográfico establece lo siguiente [6]:

- **Rompimiento total:** los criptoanlistas encuentran la llave k, de tal forma que obtienen $D_k(C) = M$.
- **Deducción global:** los analistas encuentran un algoritmo alternativo equivalente a $D_k(C)$, sin conocer la llave k.
- **Deducción local:** los criptoanalistas encuentran el texto en claro asociado a un texto cifrado.
- **Deducción de información:** los analistas obtienen cierta información acerca de la llave o el texto en claro.

Un algoritmo es llamado **incondicionalmente seguro** sin importar la cantidad de texto cifrado de la que disponga el equipo de criptoanlistas, no es suficiente para permitirles descifrar o recuperar el texto en claro. Todos los criptosistemas que presentamos son susceptibles a un ataque de texto cifrado que trate una por una, cualquier llave posible, verificando en cada caso si el texto llano resultante tiene algún sentido. Este tipo de ataque es del tipo de **fuerza bruta** y más adelante profundizaremos en él.

Es posible medir la complejidad de un ataque empleando diferentes parámetros:

- Por **complejidad de datos.** La cantidad de información o datos necesarios para iniciar un ataque.
- Por **complejidad de procesamiento.** El tiempo necesario para ejecutar el ataque. Esto también es llamado **factor de trabajo**.
- Por **requisitos de almacenamiento.** La cantidad de memoria necesaria para realizar el ataque.

La complejidad suele expresarse en "ordenes de magnitud". Si por ejemplo, la complejidad de procesamiento es de 2^{128}, esto significa que son necesarias 2^{128} operaciones para romper el algoritmo. Por supuesto, cada una de estas operaciones consume tiempo. Pero, para este caso, supongamos que disponemos de suficiente rapidez de cómputo como para realizar un millón de operaciones por segundo en una máquina y que disponemos de un conjunto de éstas trabajando en paralelo. Un ataque tardaría alrededor de 10^{19} años en recobrar la llave de encripción.

Mientras que la complejidad de un ataque es constante, a menos que se encuentre un mejor ataque, la potencia de cómputo ha tenido un gran avance durante las últimas décadas, el cual continúa. De hecho esta evolución es descrita por la ley de More [9], la cuál dice:

Cita 3. Ley de More

La complejidad de los circuitos integrados se duplicaría cada año con una reducción de costo conmensurable. Más específicamente: El número de transistores en un chip se duplica cada dos años lo que implica una disminución de los costos, puesto que los componentes y los ingredientes de las plataformas con base de silicio crecen en desempeño y se vuelven exponencialmente más económicos de producir, y por lo tanto más abundantes.

E.G. Moore, *Cramming more components into integrated circuits*, *Electronics*, V. 38 (8) 1965.

El cómputo en paralelo ofrece una opción ideal para muchos ataques criptoanalíticos, ya que las tareas pueden repartirse en una gran cantidad de procesos y ninguno de estos necesita interactuar con el otro. Los buenos criptosistemas son diseñados para que sean imposibles de romper incluso con la potencia de cómputo que se espera que evolucione muchos años en el futuro.

Para ilustrar algunas de las ideas de este capítulo, plantearemos un ataque sencillo que busca descifrar un mensaje, para lo cual es necesario "adivinar" la llave correcta de un cifrado de sustitución. Esta llave consiste en una serie de dígitos, cuya longitud también se desconoce, por lo cual un ataque puede consistir en generar todas las llaves posibles, aplicar cada una de estas llaves al proceso de descifrado y comprobar que se ha obtenido un

mensaje claro, para lo cual se buscan las palabras de un diccionario dentro del mensaje obtenido. En caso de que ninguna palabra del diccionario esté contenida en el mensaje, se prueba con la llave siguiente.

A continuación se incluye el programa *Ataque de Fuerza Bruta*, que ilustra este ataque encriptando el mensaje "ESTEESUNMENSAJEHISTORICO" con un cifrado de sustitución (que se discute en el siguiente capítulo) y una llave secreta. El ataque encuentra la llave correcta para descifrar el mensaje, generando llaves de forma aleatoria y probándolas comparando el mensaje con un diccionario. Se considera que se ha roto el cifrado cuando una palabra del diccionario está contenida en el mensaje descifrado.

Cabe mencionar que existe la posibilidad de que se encuentre una palabra del diccionario aún cuando el mensaje obtenido no tenga sentido, de tal manera que resulta necesaria la intervención humana para validar el descifrado y hacer otro ataque hasta encontrar el mensaje en claro.

Algoritmo 1. Ataque de fuerza bruta a un cifrado de sustitución.

El programa genera llaves aleatorias para tratar de descifrar el criptograma, prueba cada una de ellas verificando si alguna palabra del diccionario está contenida en el mensaje obtenido. Se considera que se ha roto el cifrado cuando se encuentra que una palabra del pequeño diccionario propuesto está contenida en el mensaje. Asimismo muestra la cantidad de operaciones necesarias para encontrar la llave correcta.

Código *Python* 1. Ataque de fuerza bruta a un cifrado de sustitución

```
from random import randint
def ordN(c):
    if c.isdigit(): return ord(c)+43
    return ord(c)
def chrN(i):
    if i>90: return chr(i-43)
    return chr(i)
def Cifra(C,K):
    return ''.join([chrN(65+(ordN(m)-65+K[i%len(K)])%36)
        for i,m in enumerate(C)])
def Descifra(E,K):
    return ''.join([chrN(65+(ordN(m)-65-K[i%len(K)])%36)
        for i,m in enumerate(E)])

Dicc = ['Hola','perro','casa','Mensaje','Guerra','termino']

M = 'ESTEESUNMENSAJEHISTORICO'
```

```
Lk = 7
KC = [randint(0,9) for i in range(Lk)]
Z = Cifra(M,KC); cT=0
for L in range(2,10):
    for k in range(10**L):
        K = [int(m) for m in ('0'*(L-len(str(k)))+str(k))] #Genera Llave
        D = Descifra(Z,K)
        logrado = False
        for m in Dicc:
            cT+=1
            if D.find(m.upper())!=-1: logrado=True
        if logrado: break
    if logrado:
        print('Cifrado:',Z)
        print('Descifrado por FB:',D)
        print('Llave encontrada:',K)
        print('Operaciones:',cT)
        break
    else:
        print('Fallo el ataque por FB para llave de%d digitos'% L)
```

Ejemplo de salida:

```
Fallo el ataque por FB para llave de 2 digitos
Fallo el ataque por FB para llave de 3 digitos
Fallo el ataque por FB para llave de 4 digitos
Fallo el ataque por FB para llave de 5 digitos
Fallo el ataque por FB para llave de 6 digitos
Cifrado: ET1KMU1NNMTOCQEIQY1QYIDW
Descifrado por FB: ESTEESUNMENSAJEHISTORICO
Llave encontrada: [0, 1, 8, 6, 8, 2, 7]
Operaciones: 7787568
```

Por razones técnicas se omiten los acentos en la salida de los programas. En su caso, el texto introducido por el usuario, se muestra entre corchetes.

Capítulo 2

Técnicas criptográficas

La palabra criptografía proviene del griego *criptos* u oculto y *grafé* o escritura, puede entenderse como *escritura oculta*. El propósito fundamental de ésta consiste en alterar las representaciones lingüísticas de ciertos mensajes para hacerlos ininteligibles a receptores no autorizados. Actualmente el término criptografía se refiere a un conglomerado de técnicas que tratan sobre la protección de la información, es el estudio de técnicas matemáticas y computacionales relacionadas con aspectos de seguridad de la información tales como: confidencialidad, autenticación, suplantación y no rechazo. El estudio de la criptografía integra conocimientos de diferentes áreas como la teoría de la información, las matemáticas y la programación. La criptografía es el estudio y la aplicación de dicho conjunto de conocimientos al problema fundamental de cómo establecer comunicaciones seguras entre dos o más entes, de tal manera que se garantice un alto nivel de confidencialidad, integridad y autenticidad en los datos intercambiados.

El empleo de técnicas criptográficas para ocultar el contenido de ciertos mensajes abarca más de 4 mil años de la historia humana. La criptografía ha sido utilizada por militares, comerciantes, servicios diplomáticos y, en general, por los gobiernos para asegurar sus comunicaciones y así proteger su información confidencial. En la primera parte de este capítulo presentaremos un conjunto de técnicas criptográficas clásicas como la esteganografía, los cifrados de sustitución y transposición, las máquinas de rotor, como la Enigma alemana, y las pistas únicas. En la segunda parte estudiaremos los dos algoritmos de cifrado más famosos de la era informática, los criptosistemas DES (*Data Encription Standard*) de llave privada y RSA (Rivest, Shamir y Adleman) de llave pública. Al final presentaremos un cifrado de *grado militar* llamado Solitario.

Uno de los estudios más completos sobre el empleo de técnicas criptográficas clásicas se debe a D. Khan [10]. En su libro *The Codebrakers*, el autor narra la historia de la criptografía y sus aplicaciones, desde sus inicios hace más de cuatro mil años, hasta la mitad del siglo XX en que jugó un papel crucial en las dos guerras mundiales. Khan expone las técnicas y el contexto en el cual fueron empleadas, por lo que se considera un referente para el estudio de las técnicas clásicas. Otras referencias indispensables para el estudio de algunas técnicas clásicas son los libros *Codes, ciphers and secret writing* de Martin Gardner [11] y *Cryptanalysis. A study of ciphers and thier solution* de Helen Gaines [12], así como *Secret and urgent, the Story of Codes & Ciphers* de Fletcher Pratt [13]. A continuación presentaremos algunas de las técnicas clásicas discutidas en detalle y contextualizadas históricamente por los autores mencionados.

2.1. Estenografía

Esta técnica sirve para esconder mensajes en otros mensajes, de tal forma que la existencia del mensaje secreto está encubierta. El emisor escribe un mensaje inocuo y de alguna forma encubre el mensaje secreto en el mismo medio en el que va dicho mensaje. A lo largo de la **h**istoria se han utilizado una gran cantidad de trucos como el empl**e**o de tinta invisible, puntos o marcas muy pequeñas sobre los caracteres del mensaje inocuo, mínimas diferencias entre la forma de escribir di**ch**os caracteres, plantillas que cubren el mensaje excepto algunos caracteres, etc.

En la actualidad esta técnica continúa en uso, la idea es la misma pero varían las formas. Se esc**o**nden mensajes secretos en archivos de imágenes. Reemplazando, por ejemplo, el bit menos significativo de cada byte que forma cierta imagen por un bit del mensaje secreto, de tal forma que la imagen no cambia apreciablem**e**nte.

Otra forma i**n**teresante de esta técnica es la llamada *función imitadora* [14], diseñada para ocultar **m**ensajes. La función modifica el m**e**nsaje que se quiere esconder dándole un perfil estadístico que se asemeja a alguna otra cosa como un anuncio clasificado de periódico, una sesión de chateo, un grupo de noticias en internet, etc. Este tipo específico de esteganografía está diseñada para dificultar la tarea de las computadoras que algunos gobiernos emplean para e**x**plorar el **i**nternet en bus**c**a de mensajes que les s**o**n de interés.

Los autores escondieron un mensaje en los párrafos anteriores marcando algunas letras en el texto, invitamos al lector a encontrarlo.

La estenografía sobre imágenes se basa en el hecho de que una imagen digital está formada por una cierta cantidad de pixeles distribuidos en una matriz. Éstos son unidades de información que contienen el color correspondiente a ese punto de la imagen. Actualmente una cámara fotográfica digital genera imágenes de 12 millones de pixeles, cada uno definido por una tripleta formada por enteros en el intervalo [0,255]. Cada elemento de la tripleta, corresponde a la intensidad de uno de los colores rojo (*red*), verde (*green*) y azul (*blue*). Este sistema es llamado RGB por sus siglas en inglés.

Dada la enorme cantidad de información contenida en una imagen digital (o mapa de bits), podemos pensar en esconder información en ella; una posibilidad sería alterar ligeramente el color de un pixel para almacenar una letra. Si consideramos las combinaciones que se pueden lograr en la tripleta, obtenemos $256^3 = 16,777,216$ y dado que sólo es necesario almacenar un número en el intervalo entero [0,25], por las 26 letras del alfabeto, es muy factible que un cambio pase inadvertido.

Ahora, si alteramos directamente el color de un pixel sumando o restando cantidades en cada una de las entradas de la tripleta, por ejemplo un número entero en el intervalo [0,8], necesitaremos la imagen original para comparar y recuperar la información. A continuación presentamos un algoritmo de cifrado y un programa que no requiere de la imagen original para recuperar la información oculta estenográficamente.

Algoritmo 2. Estenografía en imágenes digitales

Una imagen digital, o mapa de bits, es un arreglo bidimensional de pixeles y puede verse como una matriz. Un pixel, elemento de dicha matriz, es una tripleta de enteros en el intervalo [0, 255] que define su color. Tomemos un pixel de alguna imagen, correspondiente a la entrada con renglón r y columna c de la matriz. Calculemos el promedio de cada uno de los componentes del color de sus ocho vecinos:

$$\{R^\star, G^\star, B^\star\} = \frac{1}{8}\left\{\sum_{i=1}^{8} R_i, \sum_{i=1}^{8} G_i, \sum_{i=1}^{8} B_i\right\}$$

De momento, asignaremos el color formado por esta tripleta a dicho pixel. Tomemos la primera letra del mensaje en claro y le asignamos un número n, correspondiente a su posición en el alfabeto y lo representamos como la suma $n_1 + n_2 + n_3$, de tal forma que cada n_i sea menor que 9. Ahora podremos modificar el color, quedando de la siguiente manera:

$$\{R^\star \pm n_1, G^\star \pm n_2, B^\star \pm n_3\} = \{R, G, B\}_{\text{cifrado}}$$

El signo positivo se usa si el valor original es menor que 128 y el negativo en otro caso, de modo que el cambio no sea perceptible a simple vista en la imagen. Haremos lo correspondiente para cada letra del mensaje, empleando pixeles con la separación adecuada. Para este algoritmo, la llave serán los números r y c, en este caso $r = 300$, $c = 300$.

El siguiente programa oculta un mensaje en la imagen perro.png. Debe existir una imagen con este nombre en la misma carpeta donde se almacene el programa; se genera el archivo estenográfico de nombre perro1.png.

Código *Python* 2. Estenografía en imágenes digitales

```
from PIL import Image
def suma(a,b):
    return list(map(sum,zip(a,b)))
def codifica(n):
    if n>=18:
        return (9,9,n%9)
    elif n>9:
        return(9,n%9,0)
    else:
        return(n,0,0)
mensaje = input('Mensaje a ocultar: ').upper().replace(' ','')
referencia = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ*' archivo = 'perro.png'
img = Image.open(archivo)
pixeles=img.load()
x = y = 100
mensaje = mensaje + '*'
for letra in range(len(mensaje)):
    p=(0,0,0)
    codigo=referencia.find(mensaje[letra])
    for i in range(-1,2):
        for j in range(-1,2):
            if (i,j)!=(0,0):
                p=suma(p,pixeles[x+i+2*letra, y+j])
    if int(p[0]/8)>=127:
        r = int(p[0]/8)-codifica(codigo)[0]
    else:
        r = int(p[0]/8)+codifica(codigo)[0]
    if int(p[1]/8)>=127:
        g = int(p[1]/8)-codifica(codigo)[1]
    else:
        g = int(p[1]/8)+codifica(codigo)[1]
    if int(p[0]/8)>=127:
        b = int(p[2]/8)-codifica(codigo)[2]
    else:
        b = int(p[2]/8)+codifica(codigo)[2]
    pixeles[x+2*letra,y] = (r,g,b)
img.save('perro1.png')
print('Mensaje -', mensaje, '- incluido en imagen')
img.show()
```

Algoritmo 3. Descifrado de estenografía en imagen digital

El proceso para descifrar será localizar el pixel K con entradas r y c, con color $\{R, G, B\}_{\text{cifrado}}$, calcular el promedio de cada intensidad obteniendo $\{R^\star, G^\star, B^\star\}$. De este modo, al calcular la diferencia, podremos obtener las n_1, n_2 y n_3 originales y recuperar la letra del mensaje en claro.

Código *Python* 3. Estenografía (descifrado)

```
from PIL import Image
def suma(a,b):
    return list(map(sum,zip(a,b)))
referencia='ABCDEFGHIJKLMNOPQRSTUVWXYZ*'
archivo='perro1.png'
img = Image.open(archivo)
pixeles = img.load()
(k,x,y) = (0,100,100)
(r,g,b) = (0,0,0)
simple = ''
while referencia[r+g+b] !='*':
    p=(0,0,0)
    for i in range(-1,2):
        for j in range(-1,2):
            if (i,j)!=(0,0):
                p=suma(p,pixeles[x+i+2*k, y+j])
    r = abs(pixeles[x+2*k,y][0]-int(p[0]/8))
    g = abs(pixeles[x+2*k,y][1]-int(p[1]/8))
    b = abs(pixeles[x+2*k,y][2]-int(p[2]/8))
    k=k+1
    simple=simple + referencia[r+g+b]
print(simple)
```

Descifrar un mensaje escondido en otro, en algunas situaciones, equivale a resolver un acertijo contenido en el mensaje mismo. Un famoso acertijo o mensaje que contiene más información de la que aparenta es el siguiente:

$$\begin{array}{cccccc} & \mathrm{S} & \mathrm{E} & \mathrm{N} & \mathrm{D} \\ + & \mathrm{M} & \mathrm{O} & \mathrm{R} & \mathrm{E} \\ \hline \mathrm{M} & \mathrm{O} & \mathrm{N} & \mathrm{E} & \mathrm{Y} \end{array}$$

El mensaje en claro de este criptograma consiste en las tres cifras que dan significado a la suma, de modo que la solución consiste en encontrar los números que son representados por cada letra.

Cada letra se representa con un único valor numérico con la condición de que tanto **S** como **M** sean diferentes de cero. Aunque es posible resolver

el acertijo mediante un sencillo análisis algebraico usando lápiz y papel, una forma automatizada de hacerlo consiste en probar diferentes asignaciones de valores aleatorios de un dígito a cada una de las letras, hasta que la suma sea consistente, lo cual se logra con el siguiente programa.

Código *Python* 4. Solución del acertijo

```
from random import randint
import time
a1 = 'SEND'
a2 = 'MORE'
a3 = 'MONEY'
a0 = list(set(a1)|set(a2)|set(a3)) #Union de Conjuntos
D = {} # Diccionario vacio

A1 = A2 = A3 = 1; Cont = 0
ahora = time.time()
while A1+A2!=A3:
    B1 = list(range(10))
    B2 = list(range(1,10))
    for i in a0:
        D[i]= choice(B2) if (i=='M' or i=='S') else choice(B1)
        if D[i] in B1: B1.remove(D[i])
        if D[i] in B2: B2.remove(D[i])
    A1=sum([(10**(len(a1)-i-1))*D[k] for i,k in enumerate(a1)])
    A2=sum([(10**(len(a2)-i-1))*D[k] for i,k in enumerate(a2)])
    A3=sum([(10**(len(a3)-i-1))*D[k] for i,k in enumerate(a3)])
    Cont+=1
tiempo = time.time()-ahora
print('%d operaciones, en%.2f segundos.'% (Cont,tiempo))
print('%d operaciones por segundo.'% (Cont/tiempo))

for d in D: print('%s =%d'% (d,D[d]))
print()
print('%d\n+%d\n-------\n%d'% (A1,A2,A3))
```

La solución encontrada por el programa es la siguiente:

$$\begin{array}{cccccc} & 9 & 5 & 6 & 7 \\ + & 1 & 0 & 8 & 5 \\ \hline 1 & 0 & 6 & 5 & 2 \end{array}$$

Lo cual corresponde a la siguiente asignación:

S	E	N	D	M	O	R	Y
9	5	6	7	1	0	8	2

Ejemplo de salida:

```
1099181 operaciones, en 7.41 segundos.
148315 operaciones por segundo.
Y = 2
S = 9
M = 1
R = 8
D = 7
O = 0
E = 5
N = 6

  9567
+ 1085
-------
 10652
```

Por razones técnicas se omiten los acentos en la salida de los programas. En su caso, el texto introducido por el usuario, se muestra entre corchetes.

El encontrar esta solución tomó 7 segundos, aproximadamente, en una ejecución particular del programa, y fue necesario generar y probar 1,099,181 asignaciones.

2.2. Cifrados de sustitución y transposición

Antes de la existencia de las computadoras, la criptografía empleaba algoritmos basados en operar sobre caracteres, generalmente las letras de un alfabeto empleadas para escribir el mensaje. Estos algoritmos sustituían unos caracteres por otros o bien los transponían, es decir, cambiaban su posición. Los mejores algoritmos hacían ambas cosas y varias veces.

Actualmente las cosas son más complejas, pero el enfoque es en esencia el mismo. Es decir, el principal cambio ha sido que los algoritmos trabajan ahora sobre bits en lugar de sobre caracteres. Lo cual se puede considerar como un cambio en el tamaño del alfabeto, se pasó de un alfabeto de 26 letras a uno de dos elementos. Los mejores algoritmos continúan combinando elementos de sustitución y de transposición. A continuación se expondrán algunos cifrados de este tipo.

2.3. Cifrados de sustitución

En esta forma de cifrado, cada carácter en el texto llano es sustituido, bajo algún procedimiento, por otro carácter en el texto cifrado. El receptor

del texto codificado invierte la sustitución y recupera el texto en claro. En la criptografía clásica existen cuatro tipos de cifrado de sustitución:

El **cifrado de sustitución simple o monoalfabético**, en el que cada caracter del texto en claro es reemplazado por otro en el texto cifrado de forma biunívoca.

Uno de los cifrados más antiguos del cual se tiene registro es el cifrado del César, en el cual cada carácter del texto en claro es reemplazado por el respectivo carácter localizado tres lugares a la derecha módulo 26. Este cifrado es de sustitución simple, ya que el alfabeto del texto cifrado es sólo una rotación del alfabeto del texto en claro y no una permutación arbitraria[1].

Por ejemplo, en la tabla siguiente se muestran sobre color amarillo las letras por las cuales se sustituyen las del texto claro (verde) para obtener el cifrado. En este caso sólo se ha desplazado el alfabeto 5 lugares.

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

Los cifrados de sustitución simple pueden ser rotos con facilidad, ya que no esconden las frecuencias con que aparecen las diferentes letras del texto en claro, de modo que son sensibles a ataques de Fuerza Bruta como los que se discutieron previamente.

Ejemplo 2. Un criptograma empleando el cifrado del César.

Encriptar la palabra HOMERO. usando el método César. Usando la tabla anterior la palabra homero se codifica como la siguiente lista de números:

$$7\ 14\ 12\ 4\ 17\ 14$$

Usando el algoritmo de encriptación, considerando la clave = 3:

$$C \equiv (T + 3) \bmod 25$$

tenemos la siguiente lista de números:

$$10\ 17\ 15\ 7\ 20\ 17$$

[1] ¿Recuerdan a la computadora de "Odisea del Espacio", HAL 9000? ¿qué se obtiene si se desplaza una letra en el abecedario?

usando la tabla tenemos:

K R P H U R

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

Código *Python* 5. Cifrado del César

```
claro = input('Mensaje a cifrar? ')
clave = int(input('Clave? '))
tabla = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
codificado = ''
claro = claro.replace(' ','') # Se eliminan los espacios
claro = claro.upper() # Se pasa a mayusculas
for m in claro:
    codificado = codificado+tabla[(tabla.find(m)+clave)%26]

tabla_cifrado = ''.join([tabla[(tabla.find(m)+clave)%26] for m in tabla])

print('Tabla de cifrado:')

print(tabla)
print(tabla_cifrado)

print('Mensaje en claro:',claro)
print('Mensaje cifrado:',codificado)
```

Ejemplo de salida:

```
Mensaje a cifrar? [La guerra continua]
Clave? [7]
Tabla de cifrado:
ABCDEFGHIJKLMNOPQRSTUVWXYZ
HIJKLMNOPQRSTUVWXYZABCDEFG
Mensaje en claro: LAGUERRACONTINUA
Mensaje cifrado: SHNBLYYHJVUAPUBH
```

Por razones técnicas se omiten los acentos en la salida de los programas. En su caso, el texto introducido por el usuario, se muestra entre corchetes.

El **cifrado de sustitución homofónica** es como el cifrado de sustitución simple, excepto porque cada carácter del texto llano puede ser mapeado a más de un carácter en el texto cifrado. Por ejemplo, la "A" podrían corresponderle los valores "G", "K" y "U".

De acuerdo a Khan[10], los cifrados de sustitución homofónica fueron empleados en fechas tan tempranas como 1401 por el Duque de Mantua y son mucho más complicados de romper que los de sustitución simple, aunque no obscurecen del todo las propiedades estadísticas del lenguaje del texto en claro. Es suficiente un ataque de texto llano conocido para romper el cifrado. Con un ataque solamente de texto cifrado, es ligeramente más difícil, pero toma poco tiempo romperlo empleando algún programa diseñado para eso, como el programa Ataque de Fuerza Bruta antes mencionado.

El **cifrado de sustitución poligramática**, en el cual bloques de caracteres en el texto llano son encriptados por grupos. Ejemplos de estos son el cifrado *Playfair*, inventado en 1854 y empleado por los británicos durante la Primer Guerra Mundial, y que consistía en cifrar por pares de letras [10, 11].

Algoritmo 4. Cifrado *Playfair*

La llave consiste en una matriz de 4×8 que contiene los caracteres correspondientes a las 26 letras del alfabeto más los números del 2 al 7 colocados aleatoriamente, por ejemplo:

D	J	H	S	Q	M	F	T
Y	4	E	V	3	Z	C	O
I	N	W	5	A	L	K	G
6	U	X	R	7	2	P	B

Se separa el texto en claro en pares de letras, incluyendo una X en el caso de que dos letras iguales se encuentren en un mismo par. Si el numero de letras es impar, se incluye una X al final para completar el par. Ahora:

1. Si ambas letras del par, se encuentran en el mismo renglón, se sustituirá cada letra por la que se encuentra en la columna de la derecha (o la de la extrema izquierda, si se trata de la última letra del renglón). Por ejemplo, el par VE, se sustituirá por 3V.

2. Si ambas letras del par, se encuentran en la misma columna, se sustituirá cada letra por la que se encuentra en el renglón de abajo (o la del primer renglón si se trata de la última letra de la columna). Por ejemplo, el par SR, se sustituirá por VS.

3. Si cada letra se encuentra en un renglón y columna diferente, la primera letra del par se sustituirá por la letra que corresponde al mismo renglón de ésta y a la columna de la segunda. La segunda letra del par se sustituirá por la letra que corresponde al mismo renglón de ésta y a la columna de la primera. Por ejemplo, el par RO, se sustituirá por BV.

Código *Python* 6. Cifrado *Playfair*

```
from random import shuffle
def Pos(t,N):
    for r,n in enumerate(N):
        if t in n:
            return([r,n.index(t)])
def GeneraLLave():
    A = [chr(a) for a in range(65,91)] + [str(a) for a in range(2,8)]
    shuffle(A)
    return [[A.pop() for i in range(8)] for j in range(4)]
def Muestra(H):
    for m in M:
        for a in m:
            print(a,end = ' ')
        print()
    print()
def Acondiciona(T):
    R = [' ','.']
    for r in R: T = T.replace(r,'')
    p = ''; B = ''
    for t in T:
        p = p + t
        if len(p)==2:
            if p[0]!=p[1]:
                B = B + p
            else:
                B = B + p[0]+'X'+p[1]
            p = ''
    B = B + p
    if len(B)%2: B = B + 'X'
    return B
def Separa(T):
    return [[T[i],T[i+1]] for i in range(0,len(T)-1,2)]
def Encripta(T):
    AC = Acondiciona(T)
    R = Separa(AC)
    TE = ''
    for par in R:
        [r1,c1] = Pos(par[0],M)
        [r2,c2] = Pos(par[1],M)
```

```
        if r1==r2:
            k1 = M[r1][(c1+1)%8]
            k2 = M[r1][(c2+1)%8]
        elif c1==c2:
            k1 = M[(r1+1)%4][c1]
            k2 = M[(r2+1)%4][c1]
        else:
            k1 = M[r1][c2]
            k2 = M[r2][c1]
        TE = TE + (k1+k2 + ' ')
    return TE
def desEncripta(T):
    AC = Acondiciona(T)
    R = Separa(AC)
    TE = ''
    for par in R:
        [r1,c1] = Pos(par[0],M)
        [r2,c2] = Pos(par[1],M)
        if r1==r2:
            k1 = M[r1][(c1-1)%8]
            k2 = M[r1][(c2-1)%8]
        elif c1==c2:
            k1 = M[(r1-1)%4][c1]
            k2 = M[(r2-1)%4][c1]
        else:
            k1 = M[r1][c2]
            k2 = M[r2][c1]
        TE = TE + (k1+k2 + ' ')
    return TE
TC = input('Mensaje: ').upper()
print()
M = GeneraLLave()
Muestra(M)
TE = Encripta(TC)
print('Cifrado: ',TE)
TC = desEncripta(TE)
print('Descifrado:',TC)
```

Ejemplo de salida:

```
Mensaje: MAS RARO QUE UN PERRO VERDE

 D  J  H  S  Q  M  F  T
 Y  4  E  V  3  Z  C  O
 I  N  W  5  A  L  K  G
 6  U  X  R  7  2  P  B

Cifrado:    QL VS 57 3T X4 JU XC 7R BV 3V 6S WH
Descifrado: MA SR AR OQ UE UN PE RX RO VE RD EX
```

Por razones técnicas se omiten los acentos en la salida de los programas. En su caso, el texto introducido por el usuario, se muestra entre corchetes.

El **cifrado de sustitución polialfabética** es una secuencia de múltiples cifrados de sustitución simple. Por ejemplo, puede haber cinco diferentes cifrados simples de sustitución, pero uno de ellos en particular usa cambios de posición de cada carácter del texto en claro. El cifrado de sustitución polialfabética fue inventado en 1568 por Leon Battista. Fue empleado en la guerra civil del siglo XIX en E.U.A. [10]. Actualmente, este cifrado puede ser roto con cierta facilidad con la ayuda de computadoras. Otros ejemplos de este tipo de cifrado son el *Vigenère*, públicado en 1586, y el Beufort [10, 11].

El cifrado *Vigenère* es un ejemplo de cifrado polialfabético. La clave K está constituida por una secuencia de símbolos de longitud d, sin repeticiones:

$$K = \{k_0, k_1, \ldots, k_{d-1}\}$$

y emplea la siguiente congruencia lineal:

$$E_k(m_i) = (m_i + k_{i(\text{mod } d)}) \text{ mod } n$$

siendo m_i el $i-$ésimo símbolo del texto claro y n la longitud del alfabeto.

Ejemplo 3. Criptograma con cifrado de *Vigenère*

Cifrar el texto claro CURVE, usando el algoritmo *Vigenère*, con la clave SUN ($d = 3$).

$$\begin{aligned}
E(C) = E(2) &= (2 + (k_{0 \text{ mod } 3} = \text{S}) = 18) \text{ mod } 26 &&= 20(\text{U}) \\
E(U) = E(20) &= (20 + (k_{1 \text{ mod } 3} = \text{U}) = 20) \text{ mod } 26 &&= 14(\text{O}) \\
E(R) = E(17) &= (17 + (k_{2 \text{ mod } 3} = \text{N}) = 13) \text{ mod } 26 &&= 4(\text{E}) \\
E(V) = E(21) &= (21 + (k_{3 \text{ mod } 3} = \text{S}) = 18) \text{ mod } 26 &&= 13(\text{N}) \\
E(E) = E(4) &= (4 + (k_{4 \text{ mod } 3} = \text{U}) = 20) \text{ mod } 26 &&= 24(\text{Y})
\end{aligned}$$

obteniendo así el texto cifrado: UOENY.

Código *Python* 7. Cifrado *Vigenère*

```
claro = input('Mensaje a cifrar? ')
clave = input('Clave? ')
tabla = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
codificado = ''
claro = claro.replace(' ','') # Se eliminan los espacios
claro = claro.upper()         # Se pasa a mayusculas
clave = clave.upper()         # Se pasa a mayusculas
d = len(clave)
i = 0
for m in claro:
     codificado = codificado +
         tabla[(tabla.find(m)+(tabla.find(clave[i%d])))%26]
     i+=1
print('Texto en claro: ', claro)
print('Texto cifrado: ', codificado)
```

Ejemplo de salida:

```
Mensaje a cifrar? [Quien es Vigenere]
Clave? [azul]
Texto en claro:   QUIENESVIGENERE
Texto cifrado:    QTCPNDMGIFYYEQY
```

Por razones técnicas se omiten los acentos en la salida de los programas. En su caso, el texto introducido por el usuario, se muestra entre corchetes.

Los cifrados de sustitución polialfabética tienen llaves de varias letras, cada una de las cuales es empleada para cifrar una letra del texto en claro de forma cíclica. Esto es, la primera llave encripta la primera letra del texto llano, la segunda llave encripta el segundo carácter y así sucesivamente. Después de que todas las llaves han sido empleadas, son recicladas. Por ejemplo, si se emplearon 20 llaves de una letra, entonces cada veinte letras del texto en claro el cifrado se realiza con la misma llave. Esto es llamado el **periodo** del cifrado. En criptografía clásica, los cifrados con periodos largos son significativamente más difíciles de romper que aquellos con periodos cortos, tal como veremos más adelante. Actualmente existen técnicas computacionales que pueden romper fácilmente cifrados de sustitución con periodos muy largos.

El cifrado con libros, en el cual un texto es empleado para encriptar el texto del mensaje, es también del tipo de sustitución polialfabética. A pesar de que el periodo de este cifrado tiene la longitud del texto o libro empleado para hacer el cifrado, también puede ser roto con cierta facilidad [10].

2.4. Cifrado de transposición

En un cifrado de transposición, el texto en claro no cambia los caracteres que lo componen, pero sí el orden en que éstos aparecen. En un cifrado de transposición simple, el texto en claro es escrito horizontalmente en una pieza de papel cuadriculado de una longitud fija, el texto cifrado se obtiene haciendo la lectura en forma vertical, y la desencripción consiste en escribir el texto cifrado verticalmente en papel cuadriculado de idéntica longitud y extraer el texto en claro leyendo horizontalmente.

Ejemplo 4. Un cifrado de Transposición

Encriptar la frase LA GUERRA TERMINO, usando el método de transposición. En este caso, la clave es el número de columnas.

L	A	G	U
E	R	R	A
T	E	R	M
I	N	O	X

Si leemos el texto por columnas, se tiene el texto cifrado:
LETIARENGRROUAMX

Código *Python* 8. Cifrado de Transposición

```
claro = input('Mensaje a cifrar? ')
columnas = int(input("Cuantas columnas? "))
codificado = ""
claro = claro.replace(' ','')
claro = claro.upper()
renglones = int(len(claro)/float(columnas))+1
papel = [['' for x in range(columnas)] for x in range(renglones)]
claro=claro + 'X' * (renglones*columnas - len(claro))
k=0
for i in range(renglones):
     for j in range(columnas):
         papel[i][j]=claro[k]
         print(papel[i][j],end = ' ')
         k+=1
     print()
for j in range(columnas):
     for i in range(renglones):
         codificado=codificado + papel[i][j]
print('Texto en claro: ' + claro)
print('Texto codificado: ' + codificado)
```

Ejemplo de salida:

```
Mensaje a cifrar? [El veloz murcielago hindu comia feliz cardillo y kiwi]
Cuantas columnas? [8]

E L V E L O Z M
U R C I E L A G
O H I N D U C O
M I A F E L I Z
C A R D I L L O
Y K I W I X X X
Texto codificado: EUOMCYLRHIAKVCIARIEINFDWLEDEIIOLULLXZACILXMGOZOX
```

Por razones técnicas se omiten los acentos en la salida de los programas. En su caso, el texto introducido por el usuario, se muestra entre corchetes.

La frase *El veloz murcielago hindu comia feliz cardillo y kiwi La ciguena tocaba el saxofon detras del palenque de paja* es un panagrama, es decir, contiene todas las letras del alfabeto.

El criptoanálisis de este tipo de cifrado es simple, ya que las letras del texto cifrado y el texto en claro son las mismas; un análisis de frecuencia sobre el texto cifrado revelaría que cada letra tiene la misma probabilidad que en el idioma en que se escribió el mensaje original. Esto es una buena pista para el criptoanalista, que puede entonces emplear una variedad de técnicas para determinar el orden correcto de las letras y descifrar el mensaje. Si el texto cifrado se vuelve a cifrar de esta manera, la seguridad se mejora; existen incluso cifrados de transposición más complejos, pero con la ayuda de las computadoras se pueden romper la mayoría de ellos.

El cifrado alemán ADFGVX, empleado durante la primera guerra mundial, es un ejemplo de cifrado de transposición combinado con uno de sustitución simple. En su tiempo fue un algoritmo muy complejo pero fue roto por el criptoanalista francés Gorges Painvin [10].

2.5. Máquinas de rotor

En los años veinte del siglo pasado fueron inventados varios dispositivos mecánicos de encripción para automatizar este tipo de procesos. La mayoría, al igual que las calculadoras mecánicas, estaba basada en el concepto de **rotor**: una rueda mecánica diseñada para realizar una sustitución general [10, 15].

Una máquina de rotor tiene un teclado y una serie de rotores e implementa una versión del cifrado de *Vigenère*. Cada rotor tiene una permutación

arbitraria del alfabeto con 26 posiciones y realiza una sustitución simple. La salida de cada rotor está conectada a la entrada del siguiente.

Por ejemplo, en una máquina con cuatro rotores, el primero sustituye, uno a uno, los caracteres del texto en claro. Cada uno de esos caracteres de la secuencia de salida es alimentado al siguiente rotor, que a su vez los sustituye de nuevo, el texto cifrado será la salida del último rotor. Al terminar, la posición de algunos rotores cambia, de tal forma que la siguiente vez las sustituciones serán diferentes.

El dispositivo de rotores más famoso es *Enigma*. La máquina fue empleada por los alemanes durante la Segunda Guerra Mundial. La idea fue concebida por A. Scherbius y A. G. Damm en Europa y patentada en E.U.A. por a A. Scherbius [10]. Los alemanes fortalecieron considerablemente el diseño básico para emplearla durante la guerra. La máquina *Enigma* alemana tenía tres rotores, que se podían escoger de un conjunto de cinco, un conmutador que permutaba levemente el texto, un "rotor reflejo" que tenía la función de hacer que cada rotor operara sobre cada letra del texto en claro dos veces, como se muestra en el esquema de la figura 2.1. Así de complicada como era *Enigma*, su cifrado fue roto durante la Segunda Guerra Mundial por un grupo de criptoanalistas polaco el cual le explicó a los británicos su ataque. Los alemanes fueron modificando *Enigma* al transcurrir la guerra y los británicos continuaron criptoanalizando estas nuevas versiones. En la figura 2.1, se presenta una fotografía de la máquina *Enigma* original [2].

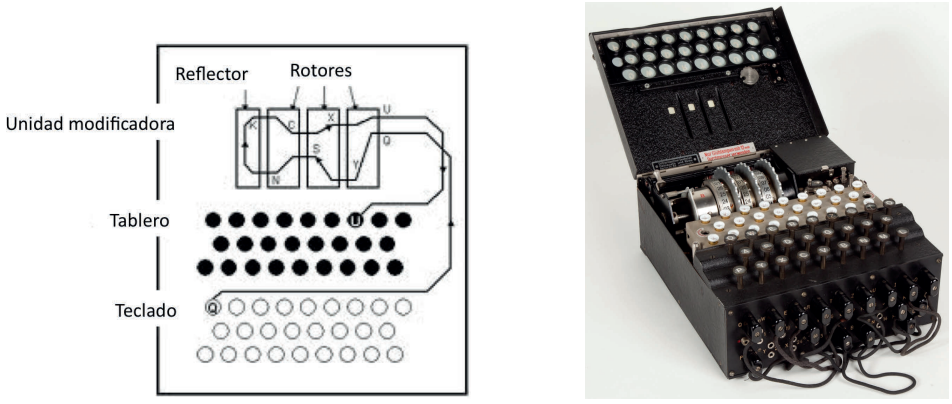


Figura 2.1: Se presenta un diagrama del funcionamiento y una fotografía de una máquina *Enigma* original.

[2] Alessandro Nassiri - Museo Nacional de Ciencia y Tecnología Leonardo da Vinci. CC BY-SA 4.0

A continuación se muestra el código que emula una máquina de rotor *Enigma* de cuatro discos, seleccionándolos de un conjunto de ocho:

Código *Python* 9. Máquina de Rotor *Enigma* (cifrado)

```
def rota(Disco,n):
    if n<0: n = len(Disco)+n
    for i in range(n):
        B = [d for d in Disco]
        C = [Disco[d] for d in Disco]
        C.insert(0,C.pop(-1))
        Disco = b:c for b,c in zip(B,C)
     return Disco
CD = []
A = [n for n in 'ABCDEFGHIJKLMNOPQRSTUVWXYZ']

CD.append([n for n in 'FJDPMSQGWXEUYRATIVHNKZCLBO']) # Disco 0

CD.append([n for n in 'SIBTGVQEOYUMDKRFNPJLZCWHAX']) # Disco 1

CD.append([n for n in 'SGJYLTERXUKZWQHOVPDMFBNICA']) # Disco 2

CD.append([n for n in 'EFCVAKUSMBJPGZLHIONRYWDQXT']) # Disco 3

CD.append([n for n in 'RXDCLUSQPIGTHVFYOZJAKBMNEW']) # Disco 4

CD.append([n for n in 'NAUXHFRZISWGPLBDOQKMCTYVJE']) # Disco 5

CD.append([n for n in 'KJGVLSEBDZUCRHFXTWOYPMAQIN']) # Disco 6

CD.append([n for n in 'ZRFKEQPVSBTHOMXWULGINJDCYA']) # Disco 7

SelDiscos = [2,6,3,0] # Seleccion del conjunto de discos
PosInicial = [4,1,6,1] # Seleccion de la posicion inicial de los discos
Engranes = [2,3,6,2] # Seleccion la rotacion de discos
D = [CD[i] for i in SelDiscos]
Discos = [a:d for a,d in zip(A,Di) for Di in D]
for i,r in enumerate(PosInicial):
    Discos[i] = rota(Discos[i],r)
Texto = input('Mensaje en claro: ')
Texto = Texto.replace(' ','')
Texto = Texto.upper()
En = ''
for L in list(Texto):
    for d in Discos: L = d[L]
    En = En + L
    for i,r in enumerate(Engranes):
        Discos[i] = rota(Discos[i],r)
print('Texto cifrado:',En)
```

Ejemplo de salida:

```
Mensaje en claro: [Vienen los rusos retirada]
Texto cifrado: HVJUBBYTFVYIPYUTQYVXJT
```

Por razones técnicas se omiten los acentos en la salida de los programas. En su caso, el texto introducido por el usuario, se muestra entre corchetes.

Para descifrar un mensaje con esta máquina, se debe invertir el orden de los discos, así como la configuración de los engranes, conservando la posición inicial de los discos, lo que forma parte de la llave de descifrado.

Código *Python* 10. Máquina de Rotor *Enigma* (descifrado)

```
def rota(Disco,n):
    if n<0: n = len(Disco)+n
    for i in range(n):
        B = [d for d in Disco]
        C = [Disco[d] for d in Disco]
        C.insert(0,C.pop(-1))
        Disco = b:c for b,c in zip(B,C)
    return Disco
CD = []
A = [n for n in 'ABCDEFGHIJKLMNOPQRSTUVWXYZ']
CD.append([n for n in 'FJDPMSQGWXEUYRATIVHNKZCLBO']) # Disco 0
CD.append([n for n in 'SIBTGVQEOYUMDKRFNPJLZCWHAX']) # Disco 1
CD.append([n for n in 'SGJYLTERXUKZWQHOVPDMFBNICA']) # Disco 2
CD.append([n for n in 'EFCVAKUSMBJPGZLHIONRYWDQXT']) # Disco 3
CD.append([n for n in 'RXDCLUSQPIGTHVFYOZJAKBMNEW']) # Disco 4
CD.append([n for n in 'NAUXHFRZISWGPLBDOQKMCTYVJE']) # Disco 5
CD.append([n for n in 'KJGVLSEBDZUCRHFXTWOYPMAQIN']) # Disco 6
CD.append([n for n in 'ZRFKEQPVSBTHOMXWULGINJDCYA']) # Disco 7
SelDiscos = [2,6,3,0] # Seleccion del conjunto de discos
PosInicial = [4,1,6,1] # Seleccion de la posicion inicial de los discos
Engranes = [2,3,6,2] # Seleccion la rotacion de discos
D = [CD[i] for i in SelDiscos]
DiscosR = [d:a for a,d in zip(A,Di) for Di in D]
for i,r in enumerate(PosInicial):
    DiscosR[i] = rota(DiscosR[i],-r)
Texto = input('Texto cifrado: ')
Texto = Texto.replace(' ','')
Texto = Texto.upper()
En = ''
DiscosR.reverse()
Engranes.reverse()
En = ''
for L in list(Texto):
    for d in DiscosR: L = d[L]
    En = En + L
    for i,r in enumerate(Engranes): DiscosR[i] = rota(DiscosR[i],-r)
print('Texto descifrado:',En)
```

Ejemplo de salida:

```
Texto cifrado: [HVJUBBYTFVYIPYUTQYVXJT]
Texto descifrado: VIENENLOSRUSOSRETIRADA
```

Por razones técnicas se omiten los acentos en la salida de los programas. En su caso, el texto introducido por el usuario, se muestra entre corchetes.

Lo que hace que una máquina de este tipo sea segura es la combinación de varios rotores y los engranajes que los mueven. Debido a que los rotores se mueven cada uno de diferente manera, el periodo de una máquina de n diferentes rotores es de 26^n. Para aumentar la seguridad y complicar el criptoanálisis, algunas máquinas de rotores incluyen diferentes posiciones en cada rotor.

2.6. Pistas únicas

Desde el punto de vista clásico, este cifrado consiste en disponer de un conjunto grande de letras generadas al azar, que constituyen una llave que se empleará una sola vez, escritas sobre hojas de papel en un cuaderno, llamado a veces cuaderno de claves. Cada una de estas hojas se llama pista. En su forma original, concebida en 1917 por J. Mauborgen, un coronel del ejercito de E.U.A., y G. Vernam un investigador de ATT, lo que se tenía era una cinta con estas características para los operadores de teletipos [10]. El emisor empleaba cada una de las letras que formaban la llave para cifrar cada uno de los caracteres del texto llano. Es decir, la encripción es la adición módulo 26 del carácter del texto en claro y el carácter correspondiente de la pista o llave.

En este cifrado cada una de las letras que forman la llave es usada una sola vez, para cifrar únicamente un mensaje. El emisor cifra el mensaje y destruye las paginas o pistas empleadas. El receptor tiene un cuaderno de claves idéntico, con las mismas pistas y emplea cada uno de los caracteres de la llave respectiva para descrifar cada letra del texto cifrado. El receptor destruye las pistas empleadas después de desencriptar el mensaje. Para cifrar un nuevo mensaje se emplean letras nuevas de la llave, es decir pistas que no han sido empleadas. Si suponemos que ningún intruso o adversario puede ganar acceso a las pistas empleadas en el cifrado, el esquema es desde el punto de vista del criptoanálisis perfectamente seguro: Un mensaje cifrado determinado puede corresponder a cualquier texto en claro del mismo ta-

maño ya que cada secuencia de caracteres de la llave es igualmente probable, por lo que no aporta información para criptoanalizarlo.

Ya que cada texto llano es igualmente posible, no existe forma de que el criptoanalista determine cuál de los mensajes de texto en claro es el correcto. Lo importante es que una secuencia aleatoria de caracteres de llave a la cuál se le suma el mensaje de texto llano, produce un texto cifrado completamente aleatorio y no importa la cantidad de poder de cómputo de que se disponga, esto no cambia.

Cualquier ataque con algún sentido que se intente contra esta forma de cifrado se dirigirá contra el método empleado para generar las letras de la llave. Para la seguridad de este esquema de cifrado es también sumamente importante que nunca se empleé más de una vez la secuencia de caracteres de la llave.

La idea de este tipo de cifrado clásico puede extenderse a los datos binarios, en lugar de pistas que consisten de letras, se emplearán pistas de bits y la operación de sumar al texto en claro la pista única se sustituirá por la operación binaria XOR. Para descifrar se opera XOR sobre el texto cifrado pero con la pista de bits empleada para codificar.

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

Nótese que: $a \oplus a = 0$ y $a \oplus b \oplus a = a$.

Existen algunos problemas asociados a la implementación de este cifrado para datos binarios ya que los bits de la llave deben ser generados aleatoriamente y no deben ser usados de nuevo, la longitud de la secuencia de la llave debe ser igual a la longitud del mensaje. Este cifrado es muy conveniente para algunos mensajes cortos, pero es muy difícil de lograr en un canal de comunicaciones de banda ancha. Uno de los problemas tiene que ver con la cantidad de información que es necesario almacenar para guardar las llaves. Aún resolviendo estos problemas de almacenamiento y distribución, es necesario resolver el problema de la sincronía, ya que es indispensable que el emisor y el receptor estén perfectamente sincronizados. Si uno de los dos está un bit adelante, o si algunos bits son eliminados durante la sesión de transmisión, el mensaje perderá sentido. Si algunos bits son alterados durante la transmisión por algo como un ruido aleatorio, únicamente esos bits

se desencriptarán incorrectamente, y esto tiene cierta importancia ya que este esquema de cifrado no proporciona autenticidad.

El cifrado de pista única es utilizado actualmente en las comunicaciones ultra-seguras en canales de ancho de banda reducidos. Se decía que, durante la Guerra Fría, la línea segura entre el Kremlin y el Capitolio estaba encriptada de esta forma. Muchos de los mensajes de los espías soviéticos de esa época estaban encriptados con esta técnica que es completamente resistente a ataques masivos de supercomputadoras [10].

Es fundamental insistir en que las letras de la llave tienen que ser **generadas aleatoriamente** lo cuál es difícil ya que los generadores de números pseudoaleatorios que emplean las computadoras tienen propiedades deterministas y emplean un generador de números pseudoaleatorios; el cual es un algoritmo que produce una sucesión de números que es una aproximación a un conjunto de números aleatorios. Esta sucesión no es exactamente aleatoria en el sentido de que queda completamente determinada por un conjunto pequeño de valores iniciales.

La mayoría de los algoritmos generadores de números pseudoaleatorios producen sucesiones que poseen una distribución uniforme. Las clases más comunes de estos algoritmos son generadores lineales congruentes, que emplean una función como la siguiente:

$$x_{n+1} = ax_n + c \bmod m$$

El adecuado funcionamiento de este tipo de generadores depende de la elección del valor inicial o semilla x_0, el valor de las constantes a y c, así como del número m. Estos valores deben ser enteros no negativos y cumplir la condición

$$x_0, a, c < m.$$

De modo que, en realidad, éstos son generadores de números pseudoaleatorios y no son confiables para la generación de llaves criptográficas de alta seguridad, ya que siempre que se parta de la misma semilla, se obtendrá la misma secuencia de valores. Existen generadores acoplados a computadoras que basan su semilla en ruido electromagnético y poseen una excelente distribución.

2.7. Algoritmos de cifrado para sistemas informáticos

El surgimiento y la proliferación de las computadoras y las telecomunicaciones en los años 60s, 70s y 80s del siglo XX impulsó la necesidad de proteger la información digital [26, 27, 28, 29, 30, 31, 32]. En los años 70s, la compañía IBM inicia un proyecto de criptografía de llave privada el cual se convierte en el antecedente del criptosistema de clave privada DES (*Data Encription Standard*), uno de los mecanismos criptográficos más conocidos y empleados de la historia reciente. Este criptosistema es empleado actualmente en la seguridad del comercio electrónico y lo utilizan muchas instituciones financieras en todo el mundo. En los párrafos siguientes presentamos algunos aspectos del criptosistema DES.

En 1976 se pública el artículo *New Direcctions in Cryptography*, en este artículo los autores Diffie y Hellman introducen el revolucionario concepto de criptografía de llave pública y proveen de un nuevo e ingenioso método para el intercambio de llaves [17, 18]. Posteriormente, en 1978 los investigadores Rives, Shamir, y Adleman desarrollan el primer criptosistema de llave pública y proponen un esquema de firma digital [23, 24, 25]. El criptosistema RSA debe su nombre a sus creadores: Rivest, Shamir y Adleman. Este algoritmo de llave pública es en la actualidad uno de los más empleados para la encripción y firma digital. Una de las principales características de los criptosistemas de llave pública es que facilitan la distribución e intercambio de llaves entre los participantes de la comunicación segura. Este es un problema fuerte en los criptosistemas de llave privada, ya que se basa en el resguardo del secreto de las llaves. Los criptosistemas de llave pública se caracterizan por el uso de llaves diferentes; una para cifrar y otra para descifrar. El criptosistema RSA es uno de los más extendidos, su seguridad se basa en la dificultad de factorizar números grandes: el atacante se enfrentará, si quiere recuperar el texto claro a partir del criptograma y la llave pública, a un problema difícil de factorización.

El criptosistema RSA emplea un algoritmo de llave pública y puede utilizarse tanto para la encripción como para firmar digitalmente. Más adelante detallaremos el criptosistema RSA.

Aunque han transcurrido más de 40 años desde la aparición de los criptosistemas clásicos DES y RSA de la era informática, se exponen de manera extendida en este texto debido a que son el fundamento a partir del cual se desarrollan criptosistemas simétricos y de llave pública más modernos.

Criptosistema DES

El criptosistema DES es un estándar internacional y es simétrico, es decir, la misma llave se emplea para encripción y desencripción.

En 1973, el NBS (*National Bureau of Standards*) solicita públicamente propuestas de sistemas criptográficos que cumplieran con las siguientes características:

- La seguridad deberá recaer en el secreto de la clave y no en el desconocimiento del algoritmo.
- Deberá proporcionar un alto nivel de seguridad y eficiencia.
- Deberá ser adaptable a diversas aplicaciones.
- Deberá ser fácilmente adaptable a diversos dispositivos.

La respuesta fue el DEA (*Data Encryption Algorithm*) o DES que desde 1977 es de uso obligatorio en el cifrado de informaciones gubernamentales no clasificadas. Este criptosistema fue desarrollado por IBM como una variación de un criptosistema anterior llamado, LUCIFER, y posteriormente tras algunas comprobaciones llevadas a cabo por la NSA (*National Security Agency*), pasó a transformarse en el que ahora conocemos como DES, el cual es un criptosistema de llave privada de 64 bits. Cada trozo de 64 bits de los datos se desordena según un esquema fijo a partir de una permutación inicial. A continuación, se divide cada uno de los trozos en dos mitades de 32 bits, que se someten a 16 iteraciones de un algoritmo. Las claves internas se utilizan en un orden para cifrar texto y en el orden inverso para descifrarlo.

Aunque no es posible demostrar rigurosamente la debilidad del criptosistema DES, y actualmente es uno de los más utilizados. Es claro que con las computadoras actuales, una clave de 56 bits (en la que reside toda la seguridad DES) es vulnerable frente a un ataque de fuerza bruta, en el que se prueben combinaciones de esos 56 bits. Frente a esto, a finales de la década de los 80's, se desarrolló el algoritmo IDEA (*International Data Encryption Algorithm*), compatible con DES para aprovechar el gran número de equipos que utilizan este algoritmo. La robustez de este criptosistema reside en una llave más larga y en un cifrador de bloques que ejecuta operaciones complejas para evitar el éxito de un posible ataque.

El algoritmo IDEA es ampliamente aceptado en las diversas aplicaciones informáticas orientadas a la seguridad. Numerosos programas destinados a trabajar en red utilizan éste como su algoritmo principal de cifrado.

Criptosistema RSA

Como ya se mencionó una de las principales características de los criptosistemas de llave pública, es que facilitan la distribución e intercambio de llaves entre los participantes de una comunicación. Este último es un problema frecuente en los criptosistemas de llave privada, ya que se basa en el resguardo del secreto de las llaves. Los criptosistemas de llave pública se caracterizan por el uso de dos llaves diferentes: una para cifrar y otra para descifrar. El criptosistema RSA es uno de los más extendidos; su seguridad se basa en la dificultad de factorizar números grandes: el atacante se enfrentará, si quiere recuperar el texto claro a partir del criptograma y la llave pública, a un problema de factorización. A continuación expondremos el algoritmo empleado por RSA para cifrar y descifrar.

Algoritmo 5. Cifrado RSA

1. Escoger dos números primos grandes, identificados como p y q.
2. Calcular el producto $n := pq$.
3. Se calcula $\rho(n) = \rho(p)\rho(q) = (p-1)(q-1)$.
4. Se escoge un número primo e relativo con $\rho(n)$, es decir, $\text{mcd}(e, \rho(n)) = 1)$.
5. Se hacen públicos n y e pero se mantienen en secreto p, q y $\rho(n)$. La llave (pública) $K_p = (n, e)$.
6. Para calcular la llave privada, escogemos en el conjunto $\{1, 2, \ldots, \rho(n) - 1\}$ un número d, que satisface la congruencia $(e\,d) \bmod \rho(n) = 1$. La clave privada es la pareja $k_p = (d, n)$. El número d debe mantenerse en secreto.

Código *Python* 11. Generador de llaves para RSA

```
from random import randrange
def mcd(a,b):
    return a if b==0 else mcd(b,a%b)
def primorelativo(p):
    q = 0
    while mcd(p,q)!=1:
        q = randrange(100,5000)
    return q
def primo():
    n = randrange(30)
    p = 2*n**2+39
    return p
def cp(r,e):
    d = 1
    while (e*d)%r!=1:
        d=randrange(r-1)
    return d
p = primo()
q = primo()
while p==q:
    q = primo()
n = p*q
rho = (p-1)*(q-1)
e = primorelativo(rho)
d = cp(rho,e)
print('p = ', p)
print('q = ', q)
print('n = ', n)
print('rho = ',rho)
print('e = ', e)
print('d = ',d)
print('Llave pública (%d,%d)'% (n,e))
print('Llave privada (%d,%d)'% (d,n))
```

Ejemplo de salida:

```
p = 1607
q = 1721
n = 2765647
rho = 2762320
e = 3023
d = 1066367
Llave publica (2765647, 3023)
Llave privada (1066367, 2765647)
```

Por razones técnicas se omiten los acentos en la salida de los programas. En su caso, el texto introducido por el usuario, se muestra entre corchetes.

Cuando alguien quiera enviar un mensaje usando la clave pública $K_p = (n, e)$, se procede de la siguiente manera:

1. Convertir el mensaje en una secuencia de dígitos agrupada.
2. Separar la secuencia de dígitos en grupos de números a_j, formando una cifra cuyo valor sea menor que n. Así, el mensaje se convierte en una lista de números $a_1, a_2, \ldots, a_r$, esta elección no es única y se pide que ningún bloque comience con cero, para evitar ambigüedades en el desencriptamiento.
3. Usando el número e como potencia módulo n calculamos los números $b_j = a_j^e \bmod n$:

$$\begin{aligned} b_1 &= a_1^e \bmod n \\ b_2 &= a_2^e \bmod n \\ &\vdots \\ b_r &= a_r^e \bmod n \end{aligned}$$

Entonces el mensaje encriptado (criptograma) es: $b_1, b_2, \ldots, b_r$.

Código *Python* 12. Cifrado RSA

```
claro = input('Mensaje a cifrar: ')
tabla = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
codificado = ''
claro = claro.replace(' ','') # Se eliminan los espacios
claro = claro.upper()         # Se pasa a mayusculas
for m in claro:
     codificado = codificado + str(tabla.find(m)+11)
cifras = 3
if len(codificado)%3!=0: cifras = 2
print claro,':', codificado
A = [int(codificado[k:k+cifras]) for k in range(0,len(codificado),cifras)]
for i,k in enumerate(A):
     print ('a'+str(i) + ' = ' + str(k))
print 'Dame la llave pública (n,e)'
n=input('n = ')
e=input('e = ')
B=[a**e%n for a in A]
vspace0.5ex
for i,k in enumerate(B):
     print ('b'+str(i) + ' = ' + str(k))
print 'Criptograma:', ','.join(map(str,B))
```

Ejemplo de salida:

```
Mensaje a cifrar: [Todos los hombres del presidente]
TODOSLOSHOMBRESDELPRESIDENTE:
3025142529222529182523122815291415222628152919141524 3015
Dame la llave publica (n,e)
n = [2765647]
e = [3023]
Criptograma:
1977108,2698124,1865328,2698124,2289346,1760369,2698124,2289346,2010657,
2698124,2474525,1502956,746285,1857539,2289346,1865328,1857539,1760369,
1674167,746285,1857539,2289346,1023197,1865328,1857539,659990,1977108,
1857539
```

Por razones técnicas se omiten los acentos en la salida de los programas. En su caso, el texto introducido por el usuario, se muestra entre corchetes.

Descifrado del RSA

Para recuperar el mensaje original (texto en claro) aplicamos la siguiente regla $x \equiv b_j^d \bmod n$, entonces:

$$\begin{aligned} a_1 &\equiv b_1^d \bmod n \\ a_2 &\equiv b_2^d \bmod n \\ &\vdots \\ a_n &\equiv b_r^d \bmod n \end{aligned}$$

lo que recupera el mensaje original $a_1, a_2, \ldots, a_r$.

Código *Python* 13. Descifrado RSA

```
cripto = input('Dame las cifras del mensaje separadas por comas: ')
tabla = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
B=map(int,cripto.split(','))
print 'Dame la llave privada (d,n)'
d=input('d = ')
n=input('n = ')
A=[b**d%n for b in B]
As = ''.join(map(str,A))
An = [int(As[k:k+2]) for k in range(0,len(As),2)]
for i,k in enumerate(A):
     print ('a'+str(i) + ' = ' + str(k))
des = ''.join([tabla[i-11] for i in An])
print('Mensaje desencriptado:', des)
```

Ejemplo de salida:

```
Dame las cifras del mensaje separadas por comas:[1977108,2698124,1865328,
2698124,2289346,1760369,2698124,2289346,2010657,2698124,2474525,1502956,
746285,1857539,2289346,1865328,1857539,1760369,1674167,746285,1857539,
2289346,1023197,1865328,1857539,659990,1977108,1857539]
Dame la llave privada (d,n)
d = [1066367]
n = [2765647]
Mensaje desencriptado: TODOSLOSHOMBRESDELPRESIDENTE
```

Por razones técnicas se omiten los acentos en la salida de los programas. En su caso, el texto introducido por el usuario, se muestra entre corchetes.

Ejemplo 5. Criptograma empleando RSA

Se usará la siguiente tabla para codificar texto a números:

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

Códificación a números del mensaje GUERRA usando la tabla y sumando 11:

G	U	E	R	R	A
17	31	15	28	28	11

Partimos de dos primos, $p = 31$ y $q = 37$. Entonces

$$n = p\,q = 1147,$$

$$\rho(n) = (31-1)(37-1) = 1080,$$

podemos elegir $e = 367$, ya que $\mathrm{mcd}(367, \rho(n)) = 1$, y $d = 103$, ya que $(e\,d = 37801) \bmod (\rho(n) = 1080) = 1$
entonces:

- Clave pública $K_p = (1147, 367)$,
- Clave privada $k_p = (103, 1147)$.

Del texto en claro codificado a números obtenemos la secuencia 173215292911, podemos separarla en bloques de tres cifras:

$$a_1 = 173$$
$$a_2 = 115$$
$$a_3 = 282$$
$$a_4 = 811$$

Claculamos las b's:

$$b_1 = 173^{367} \bmod 1147 = 102$$
$$b_2 = 215^{367} \bmod 1147 = 548$$
$$b_3 = 292^{367} \bmod 1147 = 606$$
$$b_4 = 911^{367} \bmod 1147 = 625$$

Entonces, el criptograma es : $102, 548, 606, 625$.

Ejemplo 6. Decodificación del criptograma.

Las a's se pueden recuperar de la siguiente manera:

$$a_1 = b_1^{103} \bmod 1147 = 173$$
$$a_2 = b_2^{103} \bmod 1147 = 115$$
$$a_3 = b_3^{103} \bmod 1147 = 282$$
$$a_4 = b_4^{103} \bmod 1147 = 811$$

De esta manera obtenemos la secuencia de números: 173215292911 que ordenados en parejas, y relacionándolas con la tabla anterior, nos da:

17	G
32	U
15	E
29	R
29	R
11	A

Cifrado del Solitario

El algoritmo de Cifrado Solitario, fue diseñado por Bruce Schneier, para la novela *Cryptonomicón* de Neal Stephenson [19]. El cifrado Solitario basa su seguridad de la aleatoriedad de un mazo de cartas barajado. Manipulando el mazo, un emisor puede crear una cadena aleatoria de letras para cifrar el mensaje. Este cifrado puede simularse en una computadora, pero fue diseñado para ser ejecutado sin ella. El cifrado puede ser de baja tecnología, pero la intención es que su seguridad sea de grado militar. Evidentemente, nada garantiza que sea imposible encontrar un ataque ingenioso, pero el algoritmo es mejor que otros cifrados de lápiz y papel. Tiene la desventaja de no ser rápido, puede llevar toda una tarde cifrar o descifrar un mensaje razonablemente largo si no se emplea una computadora.

Cita 4. Algoritmo del Solitario.

Es un cifrado de flujo de salida-retroalimentación. La idea básica es que genera una secuencia, en ocasiones llamada *secuencia de clave*, de números entre 1 y 26. Para cifrar, es necesario generar el mismo número de letras de la secuencia de clave como letras tiene el texto claro. Luego hay que sumar ambas módulo 26, para crear el texto cifrado. Para descifrar, hay que generar la misma secuencia de clave y restar módulo 26 del texto cifrado para recuperar el texto claro.

La descripción del cifrado y descifrado sirve igual para cualquier cifra de flujo de salida-retroalimentación. Esta sección explica el funcionamiento de Solitario.

El cifrado *Solitario* genera una secuencia de clave empleando un mazo de cartas. Puede considerar un mazo de 54 cartas (recuerde los comodines) como una permutación de 54 elementos. Hay como 54!, o unas $2{,}31 \times 10^{71}$, formas posibles de ordenar el mazo. Mejor aún, hay 52 cartas en el mazo (sin los comodines) y 26 letras en el alfabeto inglés. Esta coincidencia nos será muy útil.

Para su empleo en el cifrado se necesita un mazo con las 52 cartas y dos comodines. Los comodines deben diferir de alguna forma, el mazo que estoy usando mientras escribo este texto tiene estrellas en los comodines: uno tiene una estrella pequeña y el otro una estrella grande. Digamos que uno es el comodín A y el otro el B. Por lo general, hay un elemento gráfico en los comodines que es igual pero de tamaño diferente. Que el comodín ≪B≫ sea el ≪mayor≫. Si es más fácil, puedes escribir

una ≪A≫ y una ≪B≫ en los dos comodines, pero recuerda que luego tendrás que explicárselo a la policía secreta si te pillan. Para inicializar el mazo, pon el mazo sobre la mano mirando hacia arriba. Luego, dispón las cartas en la configuración inicial que es la clave (hablaré de la clave más tarde, pero es diferente de la secuencia de clave). Ahora estás listo para producir la secuencia de clave.

- Generar las letras de la secuencia de clave

  1. Encuentra el comodín A. Desplázalo una carta hacia abajo (es decir, intercámbialo con la carta que tenga debajo). Si el comodín se encuentra al fondo del mazo, colócalo sobre la carta superior.
  2. Encuentra el comodín B. Desplázalo dos cartas hacia abajo. Si el comodín se encuentra en el fondo del mazo, desplázalo justo debajo de la segunda carta. Si el comodín está justo encima de la última carta, desplázalo justo debajo de la primera carta (básicamente, debes asumir que el mazo es un bucle completo).

     Es importante realizar estos pasos en orden. Es tentador gandulear y limitarse a desplazar los comodines en el orden en que te los encuentras. Se puede, a menos que estén muy cerca el uno del otro.

     Así que si el mazo tiene este aspecto antes del paso 1:

     3 A B 8 9

     al final del paso 2 debería quedar así: 3 A 8 B 9

     Si tienes cualquier duda, recuerda desplazar el comodín A antes que el comodín B. Y ten cuidado cuando los comodines se encuentren al fondo del mazo.
  3. Realiza un corte triple. Es decir, intercambia las cartas sobre el primer comodín con las cartas bajo el segundo comodín. Si el mazo tiene este aspecto:

     2 4 6 B 4 8 7 1 A 3 9

     entonces, después del corte triple quedará así:

     3 9 B 4 8 7 1 A 2 4 6

«Primer» y «Segundo» comodines se refiere al comodín que esté más cerca, y más lejos, de la parte alta del mazo. Ignora la «A» y la «B» en este paso.

Recuerda que los comodines y las cartas que hay entre ellos no se mueven; las otras cartas se desplazan a su alrededor. Es fácil de hacer entre las manos. Si no hay cartas en una de las tres secciones (ya sea porque los comodines están juntos o uno en la parte alta o en la parte baja), simplemente trata esa sección como vacía y desplázala igualmente.

4. Realiza un corte contando. Mira la carta del fondo. Conviértela en un número entre 1 y 53. (Emplea el orden de palos del bridge: tréboles, diamantes, corazones y picas. Si la carta es de ♣, entonces tiene el valor que muestra. Si la carta es de ♢, entonces es su valor más 13. Si es de *heartsuit*, es su valor más 26. Si es de ♠, es su valor más 39. Cualquiera de los comodines vale 53.) Cuenta ese número hacia abajo desde la carta más alta (yo normalmente cuento desde 1 hasta 13 una y otra vez si es preciso; es más simple que contar en secuencia hasta números altos.) Corta después de la carta hasta la que has llegado, dejando la carta del fondo abajo del todo. Si el mazo tenía este aspecto:

   7 ... cartas ... 4, 5 ... cartas ... 8, 9

   y la novena carta era el 4, el corte daría como resultado:

   5 ... cartas ... 8, 7 ... cartas ... 4, 9

   La razón por la que se deja la última carta en su lugar es para que este paso sea reversible. Es un detalle importante de cara al análisis matemático de su seguridad.

5. Busca la carta de salida. Mira la carta en la parte alta. Conviértela en un número entre 1 y 53, de la misma forma que antes. Cuenta hacia abajo ese número de cartas (cuenta la primera carta como número uno). Apunta en una hoja de papel la carta que encontraste después de la que contaste (si te encuentras con un comodín, no escribas nada y comienza de nuevo con el paso 1). Ésta es la primera carta de salida. Ten en cuenta que este paso no modifica el estado del mazo.

6. Convierte la carta en un número. Como antes, emplea los palos del bridge para ordenarlos. De más bajo a más alto tenemos tréboles, diamantes, corazones y picas. Por tanto, de A♣ hasta K♣ es de 1 a 13, de A♢ hasta K♢ es de 14 a 26, de A♡ hasta K♡ es de 1 a 13, y de A♠ hasta K♠ es de 14 a 26.

   Esto es Solitario. Puedes emplearlo para generar tantos números de la secuencia de clave como te sean necesarios.

   Sé que hay diferencias regionales en los mazos de cartas, dependiendo del país. En general, no importa la ordenación de palos que emplees o cómo conviertas las cartas en números. Lo que importa es que el emisor y el receptor hayan acordado las mismas reglas. Si no eres consistente no podrás comunicarte.

- Introducir la clave en el mazo

  Solitario es tan seguro como la clave. Es decir, la forma más simple de romper Solitario es descubrir la clave que los comunicantes están empleando. Si no dispones de una buena clave, el resto no importa. Aquí hay algunas sugerencias para intercambiar claves.

  1. Baraja el mazo. Una clave aleatoria es la mejor. Uno de los comunicantes puede barajar un mazo aleatoriamente y luego crear otro mazo idéntico. Uno pasa al emisor y otro al receptor. La mayoría de la gente no sabe barajar bien, así que barájalo al menos diez veces, e intenta usar un mazo con el que se haya jugado en lugar de uno nuevo, recién sacado del paquete. Recuerda conservar un mazo extra con el orden de la clave, o en caso de error nunca podrás descifrar el mensaje. También recuerda que la clave está en peligro mientras exista; la policía secreta podría encontrar el mazo y copiar el orden.
  2. Emplea una ordenación de bridge. Una descripción de manos de bridge que puedes encontrar en un periódico o un libro sobre bridge es aproximadamente una clave de 95 bits. Si los comunicantes se ponen de acuerdo en una forma de convertirlas en una ordenación del mazo y en una forma de situar los comodines (quizá después de las dos primeras car-

tas que se mencionan en la discusión del juego), puede valer. Atención: la policía secreta podría encontrar tu columna de bridge y copiar el orden. Puedes intentar establecer alguna convención repetible para la columna del bridge a emplear; por ejemplo, ≪usa la columna del bridge del periódico de tu ciudad del día en que cifraste el mensaje≫, o algo similar. O busca una serie de palabras en la web del New York Times, y emplea la columna de bridge del día del artículo que te aparece cuando buscas por esas palabras. Si se descubren las palabras, o las interceptan, parecerán una frase clave. Y elige tu propia convención; recuerda que la policía secreta también lee los libros de Neal Stephenson.

3. Utiliza una frase clave para ordenar el mazo. Este método emplea Solitario como un algoritmo para crear una ordenación inicial del mazo. Tanto el emisor como el receptor comparten la frase clave (por ejemplo, ≪CLAVE SECRETA≫). Comienza con el mazo en un orden fijo, de la carta más baja a la más alta, de la forma habitual en el bridge. Realiza la operación Solitario, pero en lugar del paso 5, realiza otro corte contado empleando el primer carácter de la frase clave (3 en este ejemplo). Recuerda colocar las cartas en la posición más alta sobre la carta más baja del mazo, como antes. Hazlo para cada uno de los caracteres. Emplea otros dos caracteres para fijar la posición de los comodines. Para que sea realmente seguro querrás usar una frase clave de al menos 80 caracteres; yo recomiendo al menos 120 caracteres.

N. Stephenson, *Criptonomicón III, Código Aretusa*, De bolsillo, 2005.

Ejemplo 7. Criptograma con el cifrado Solitario.

Para cifrar el primer mensaje Solitario mencionado en la novela de Stephenson, ≪DO NOT USE PC≫:

1. Dividir el mensaje en texto llano en grupos de cinco caracteres (no hay nada mágico en los grupos de cinco caracteres; es simplemente la tradición). Se usan equis para rellenar el último grupo. Así que si el mensaje es ≪DO NOT USE PC≫ entonces el texto llano es:

   DONOT USEPC

2. Usar Solitario para generar diez letras de la secuencia de clave (los detalles se dan después). Demos por supuesto que son:

   KDWUP ONOWT

3. Convertir el mensaje en texto llano de letras a números: A=1, B=2, etc:

   4 15 14 15 20 21 19 5 16 3

4. Convertir las letras de la secuencia de clave de forma similar:

   11 4 23 21 16 15 14 15 23 20

5. Sumar los números de la serie del texto llano a los números de la secuencia de clave, módulo 26.

   15 19 11 10 10 10 7 20 13 23

6. Convertir los números en letras:

   OSKJJ JGTMW

   Si eres realmente bueno, puedes aprender a sumar letras de cabeza, y simplemente sumar las letras del paso (1) y (2). Sólo se precisa práctica. Es fácil recordar que A+A=B; recordar que T+Q=K es más difícil.

La idea básica para descifrar es que el receptor genere la misma secuencia de clave, y luego reste las letras de la secuencia de clave de las del texto cifrado.

1. Tomar el mensaje cifrado y disponerlo en grupos de cinco caracteres (ya debería tener esta forma).

   OSKJJ JGTMW

2. Usar Solitario para generar diez letras de la secuencia de clave. Si el receptor emplea la misma clave que el remitente, las letras de la secuencia de clave serán las mismas:

   KDWUP ONOWT

3. Pasar el mensaje cifrado de letras a números:

   15 19 11 10 10 10 7 20 13 23

4. Convertir de forma similar las letras de la secuencia de clave:

   11 4 23 21 16 15 14 15 23 20

5. Restar los números de la secuencia de clave de los números del texto cifrado, módulo 26. Por ejemplo, 22-1=20,1-22=5. (Es fácil. Si el primer número es menor que el segundo, añadir 26 al primer número antes de restar. Por tanto 1-22=? se convierte en 27-22=5.)

   4 15 14 15 20 21 19 5 16 3

6. Convertir los números en letras:

   DONOT USEPC

   Descifrar es igual que cifrar, excepto que se resta la secuencia de clave del mensaje cifrado.

El siguiente código genera un archivo de texto, que emula un mazo barajado de cartas que servirá como la llave para cifrar.

Código *Python* 14. Solitario. Generador de llave de cifrado

```
from random import randint
print('Generar configuracion inicial:')
opcion=input('1. de manera aleatoria 2. introducirla manualmente ')
def Swap(x,y):
    j,k = M[x],M[y]
    M[x],M[y] = k,j
archivo=open('LlaveSolitario.txt','w')
if opcion== '1':
    M = [i+1 for i in range(54)]

    for i in range(5000):
        a = randint(0,53)
        b = randint(0,53)
        Swap(a,b)

    print('El orden del mazo es:')
    for m in M:
        archivo.write('%d\n'% m)
        print(m,end=' ')

else:
    for i in range(54):
        m=input('Carta '+str(i)+':')
        archivo.write('%d\n'% m)
archivo.close()
```

Ejemplo de salida:

```
Generar configuracion inicial:
1. de manera aleatoria 2. introducirla manualmente [1]
El orden del mazo es:
3 39 22 48 29 50 33 53 25 42 32 37 15 43 2 51 16 1 17 4 30 12 14 21 34 28
40 31 7 41 26 47 19 36 6 44 9 35 49 54 5 45 18 10 38 20 52 13 46 27 8 23
24 11
```

Por razones técnicas se omiten los acentos en la salida de los programas. En su caso, el texto introducido por el usuario, se muestra entre corchetes.

El siguiente código permite cifrar o descifrar un mensaje a partir de la llave generada por el código anterior.

Código *Python* 15. Solitario descifrado

```
def Swap(x,y):
    j,k = M[x],M[y]
    M[x],M[y] = k,j
# Las cartas son los numeros del 1 al 54
# el indice del arreglo que modela al mazo va del 0 al 53
tabla = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
opcion=input('Quieres: 1. Encriptar, 2. Descifrar ')
Texto=input('Mensaje: ')
Texto = Texto.replace(' ','') # Se eliminan los espacios
Texto = Texto.upper()         # Se pasa a mayusculas
Codificado = []
for m in Texto:
     Codificado = Codificado + [tabla.find(m)+1]
L = list(open('LlaveSolitario.txt','r'))
M=[]
for l in L:
        l = l.replace('\n','')
        M=M+[int(l)]
ComodinA = 53
ComodinB = 54
Salida=[]
while len(Salida)<len(Texto):
    Comodin1 = M.index(ComodinA) # Posicion del comodin 1
    Swap(Comodin1,(Comodin1+1)%54)
    Comodin2 = M.index(ComodinB) # Posicion del comodin 2
    Swap(Comodin2,(Comodin2+1)%54)
    if M[53]==ComodinB:
        M = [M[0]]+[M[53]]+M[1:53]
    else:
        Swap((Comodin2+1)%54,(Comodin2+2)%54)
    Comodin1=min(M.index(ComodinA),M.index(ComodinB))
    Comodin2=max(M.index(ComodinA),M.index(ComodinB))
    M = M[Comodin2+1:54]+[M[Comodin1]]+M[Comodin1+1:Comodin2]
              +[M[Comodin2]]+M[0:Comodin1]
    Ultima = M[53]
    M = M[Ultima:53]+ M[0:Ultima]+[Ultima]
    Primera = M[0]
    if Primera==54: Primera=53
    if M[Primera]!=ComodinA and M[Primera]!=ComodinB:
        Salida = Salida+[M[Primera]]
k = ''
for i,m in enumerate(Codificado):
    k = k+tabla[(m+Salida[i]-1)%26]
if opcion=='1':
    print('Cifrado:',k)
else:
    claro = ''
    for i,n in enumerate(Codificado):
        p=(n-Salida[i]-1)%26
        claro = claro + tabla[p]
    print('Descifrado:',claro)
```

Ejemplo de salida:

```
Quieres: 1. Encriptar, 2. Descifrar [1]
Texto: Todos los hombres del presidente
Cifrado: DGNONAVZCFBWZHMBFIHMVEIZADWD

Quieres: 1. Encriptar, 2. Descifrar [2]
Texto: DGNONAVZCFBWZHMBFIHMVEIZADWD
Descifrado: TODOSLOSHOMBRESDELPRESIDENTE
```

Por razones técnicas se omiten los acentos en la salida de los programas. En su caso, el texto introducido por el usuario, se muestra entre corchetes.

Capítulo 3

Protocolos de comunicaciones

En este capítulo presentaremos una visión general sobre los protocolos en criptografía y las comunicaciones seguras. Un **protocolo** es una serie de pasos que involucran a dos o más partes o participantes en un evento y está diseñado para lograr realizar una tarea determinada. Ya que es una definición importante, vamos a descomponerla en sus partes: "una serie de pasos" significa que el protocolo posee una secuencia desde el principio hasta el final. Cada uno de los pasos debe ser ejecutado en su momento, de tal forma que no pueda realizarse sin que el paso anterior esté completo o haya finalizado. Que "involucran a dos o más partes o participantes en un evento" significa que al menos dos personas son necesarias para completar el protocolo. Una persona solamente no hace un protocolo, es decir, ésta puede realizar una serie de pasos para realizar una determinada tarea, pero esto no es un protocolo. Y finalmente "está diseñado para lograr realizar una tarea determinada" significa que el protocolo debe lograr algo, completar la tarea para la que fue diseñado.

Las características fundamentales o propiedades que un protocolo debe tener son las siguientes:

1. Cada uno de los involucrados en el protocolo debe conocerlo, así como toda la serie de pasos a seguir por adelantado.
2. Cada uno de los involucrados en el protocolo debe de estar de acuerdo en seguirlo.
3. El protocolo no debe ser ambiguo, cada uno de sus pasos debe estar bien definido y no debe haber lugar a un malentendido.

4. El protocolo debe ser completo, es decir, debe existir una acción específica para cada situación posible.

Un protocolo criptográfico es simplemente un protocolo que emplea la criptografía. Las partes o los participantes pueden ser "amigos" y confiar cada uno en el otro o bien pueden ser adversarios entre sí y no confiar en lo absoluto en los otros. El protocolo involucra algún algoritmo criptográfico, pero generalmente su propósito es algo más allá de puramente la secrecía. Por ejemplo, los participantes en el protocolo podrían querer compartir partes de algo secreto para poder calcular algún valor determinado o conjuntamente generar una secuencia de números aleatorios, convencer uno a otro de su identidad, firmar simultáneamente un contrato, etc. El punto importante es que el empleo de la criptografía en un protocolo tiene la función de prevenir o detectar la presencia de alguien que secretamente "escucha" las comunicaciones legítimas entre las partes o de un impostor interesado en engañarlas.

Existen muchos ejemplos de protocolos informales en la vida diaria: el cruce de semáforos, el adquirir artículos en una tienda, identificarnos en un banco para retirar dinero, votar en unas elecciones, etc. Es frecuente que no reflexionemos sobre estos protocolos ya que están ahí desde hace mucho y aunque han ido cambiando con el tiempo todos sabemos como usarlos y han funcionado medianamente bien, salvo en algunos casos. Con la llegada y el crecimiento de Internet cada vez más la interacción entre humanos se realiza sobre una red de computadoras en lugar de "cara a cara" o de forma presencial. Es decir, nos comunicamos con frecuencia por teléfono celular, enviando mensajes de correo electrónico, en chats, blogs y toda una amplia gama de aplicaciones para comunicaciones. Y resulta que las computadoras necesitan protocolos formales para hacer las mismas cosas que las personas hacen en su interacción social de forma natural y sin pensarlo demasiado. Muchos de los protocolos "cara a cara" confían en la presencia de las personas para garantizar cierta seguridad e imparcialidad. Pero seríamos ingenuos si suponemos que todas las personas, que no conocemos y con las que interactuamos vía una red de comunicaciones informáticas, son honestas. Es inocente también suponer que todos los administradores de dichas redes son honestos, y la deshonestidad puede provocar serios daños. Al formalizar los protocolos disponemos de una herramienta que nos permite examinar las formas en que las personas o partes deshonestas pueden romper dicho protocolo y hacer trampa. Podremos entonces desarrollar protocolos que nos permitan subsanar las fallas.

Otra de las características de los protocolos es que hacen abstracción de los procesos necesarios para lograr una tarea, separándolos de los mecanismos específicos mediante los cuales se realizan. Por ejemplo, un protocolo de comunicación, como el TCP/IP, es el mismo independientemente de la plataforma donde esté implementado.

Para ayudar a mostrar los diferentes protocolos, es muy frecuente emplear una metodología que consiste en crear personajes o jugadores que desempeñan ciertos papeles en el juego. Para analizar este juego de las comunicaciones seguras, proponemos el siguiente reparto:

Florencia: la primer participante en todos los protocolos.

Francisco: el segundo participante en los protocolos.

Clemente: participante en los protocolos de tres o cuatro partes.

Ramón: participante en los protocolos de cuatro partes.

David: instruso que escucha secretamente la conversación.

Patrick: atacante activo y malicioso.

Ángel: arbitro confiable.

Claudio: guardián, encargado de cuidar de Florencia y Francisco.

Alberto: verificador.

Arturo: el investigador que demuestra la existencia de alguna irregularidad en el protocolo aportando evidencias.

Agregamos que, como regla general, Florencia iniciará todos los protocolos y Francisco le responderá, los demás actores jugaran papeles de soporte especializados, salvo Clemente y Ramón que participarán sólo en el caso de protocolos que involucren tres o cuatro partes.

3.1. Protocolos arbitrados

Un árbitro es un tercero confiable, desinteresado y que es necesario para completar un protocolo. *Desinteresado* significa que dicho arbitro no tiene intereses concedidos en el protocolo y no tiene ninguna lealtad o preferencia hacia ninguno de los participantes involucrados. *Confiable* significa que todos los involucrados en el protocolo aceptan que lo que él dice es verdad, lo que

hace es correcto y que completará su parte respectiva en el protocolo. Una de las funciones del árbitro es ayudar a completar protocolos en los cuales las partes desconfían mutuamente.

Un ejemplo de arbitro es un notario o un abogado. Pensemos que Florencia le va a vender a Francisco, un completo extraño, una casa. Debido a la cantidad de la transacción, Francisco necesita pagar con cheque, pero Florencia no tiene forma de saber si el cheque es bueno y tiene fondos, por lo que necesita asegurarse de esto de alguna forma antes de firmar y entregarle las escrituras a Francisco. Por otro lado, Francisco tampoco confía en Florencia, no sabe si su escritura es apócrifa o no, y tampoco le gustaría entregar un cheque sin recibir a cambio la escritura correcta. Una forma de resolver el problema es emplear a un notario que tenga la confianza de ambos. Con la ayuda del notario se puede emplear el siguiente protocolo para asegurar que ninguna de las partes engañará a la otra:

1. Florencia le entrega el título al notario.
2. Francisco le entrega el cheque a Florencia.
3. Florencia deposita el cheque.
4. Después de esperar un cierto periodo de tiempo para verificar que el depósito del cheque esté bien, el notario le entrega las escrituras a Francisco. En caso de que el cheque sea falso o no tenga fondos, Florencia le mostrará la evidencia de esto al notario y éste le regresará las escrituras.

El concepto del arbitraje es muy viejo y es parte de la sociedad misma. A lo largo de la historia de las relaciones humanas han existido personas que tienen la autoridad otorgada o confiada por los demás miembros para actuar con justicia y equilibrio entre las partes. Estas figuras ideales, podrían trasladarse al mundo de las redes de computadoras, sin embargo existen una serie de problemas para hacerlo:

1. Es relativamente simple encontrar y confiar en un tercero neutral si conocemos su trayectoria y podemos ver su rostro. Pero si dos partes sospechan mutuamente entre sí, es natural que sospechen de un arbitro virtual que se encuentra en algún lugar de la red.
2. La red de computadoras debe cargar con el costo del arbitraje. Así como abogados y notarios cobran por realizar su trabajo, también lo hacen estos árbitros virtuales.

3. Existe una demora inherente a cualquier protocolo arbitrado.

El árbitro debe participar en cada transacción, convirtiéndose así en un cuello de botella para todas aquellas implementaciones de protocolos a gran escala. El incrementar el número de árbitros en este tipo de implementaciones puede mitigar el problema aunque aumenta los costos. Puesto que todos los involucrados en la red confían en el arbitraje, éste representa un punto vulnerable y fácil de identificar para cualquiera que esté intentando vulnerar la red. A pesar de estos inconvenientes, los árbitros juegan un papel importante en las redes informáticas. En el caso de los protocolos que mostraremos, el papel de árbitro lo juega Ángel.

3.2. Protocolos adjudicados[1]

Debido al alto costo que tiene el uso de árbitros, los protocolos arbitrados se dividen en dos subprotocolos de menor nivel. Uno de éstos es un subprotocolo no arbitrado que se lleva a cabo cada vez que las partes necesitan completar el protocolo y cooperan para ello. El otro es un subprotocolo arbitrado que se ejecuta solamente cuando hay una disputa. Este tipo especial de árbitro es un juez, un tercero confiable y desinteresado pero que no está directamente involucrado en el protocolo. Éste es llamado únicamente en caso de que sea necesario determinar si el protocolo se realizó de manera justa. Esto guarda una fuerte analogía con la vida diaria, ya que un juez profesional sólo es necesario en caso de que haya una disputa entre las partes, a diferencia del notario o el abogado, que son árbitros que participan en los protocolos. Un juez no ve el contrato hasta que una de las partes fuerza a la otra a ir a una corte.

Este protocolo de firma de contrato puede formalizarse de la siguiente manera:

Subprotocolo no arbitrado.

1. Florencia y Francisco negocian los términos del contrato.

2. Primero Florencia y después Francisco firman el contrato.

Subprotocolo adjudicado (ejecutado solo en caso de disputa)

3. Florencia y Francisco se presentan frente al juez.

[1]Del latín *adjudicat*, protegido judicialmente.

4. Florencia presenta su evidencia.
5. Francisco Presenta su evidencia.
6. El juez aplica las reglas basandose en la evidencia y emite una sentencia.

Existen protocolos adjudicados para las comunicaciones a través de redes de computadoras. Éstos confían en que las partes serán honestas, pero si alguien tiene una duda y sospecha de un engaño, existe un conjunto de datos que son almacenados de tal forma que un tercero pueda determinar si se está haciendo trampa. En un protocolo de este tipo el juez podrá determinar también la identidad del tramposo. La función de estos protocolos no es preventiva pero permite detectar los engaños.

3.3. Protocolos autoreforzados

La idea de este tipo de protocolos es garantizar por sí mismos la imparcialidad, de tal manera que no sea necesario un árbitro para completar el protocolo y tampoco sea necesario un juez para resolver las disputas. El protocolo está construido o diseñado de forma tal que no se puedan dar las disputas; si una de las partes intenta engañar a las otras, éstas pueden detectar inmediatamente el engaño y detener el protocolo. Desafortunadamente, no existe un protocolo autorreforzado para cada situación.

3.4. Ataques contra protocolos

Un ataque criptográfico puede ser dirigido contra los algoritmos criptográficos empleados en los protocolos, contra las técnicas criptográficas usadas para implementar los algoritmos y protocolos o contra los protocolos en sí mismos. De estos tres tipos de ataque vamos a examinar, por ahora, solamente aquellos dirigidos contra los protocolos, suponiendo entonces que, tanto los algoritmos criptográficos como las técnicas son seguros.

Ataques pasivos

En este caso, alguien no involucrado en el protocolo puede espiar a algunos de los participantes. Este ataque es llamado **pasivo** debido a que el atacante no afecta el protocolo, sino que observa y trata de obtener información (como un ataque de texto cifrado). En general, es difícil detectar a los

atacantes pasivos de tal forma que es conveniente considerar su prevención dentro del protocolo. En los protocolos que estudiaremos más adelante, el papel del intruso que escucha secretamente la comunicación lo hará David.

Ataques activos

Por otro lado un atacante puede tratar de alterar el protocolo para tomar ventaja de esto. Él podría pretender ser alguien más e introducir nuevos mensajes en el protocolo, borrar mensajes existentes, sustituir unos por otros, interrumpir el canal de comunicaciones o alterar la información almacenada en la computadora. Este tipo de ataque se llama **activo** porque tiene una intervención activa en el protocolo; las formas específicas de este ataque dependen de las características de la red que se esté intentando vulnerar. El papel del atacante activo y malicioso en los protocolos que estudiaremos lo jugará Patrick.

En general, los atacantes pasivos tratan de obtener información sobre las distintas partes involucradas en el protocolo. Para lo cuál recolectan los mensajes que envían las partes para criptoanalizarlas. Por otra parte, los atacantes activos pueden tener objetivos diversos: pueden estar interesados en obtener información, en degradar la eficiencia del sistema, en corromper la información almacenada o en ganar acceso a recursos y privilegios no autorizados. Los atacantes activos son peligrosos especialmente en los protocolos en los que las partes no necesariamente confían entre sí. El atacante no tiene por qué ser un completo extraño al protocolo, podría ser un usuario legítimo del sistema, e incluso el administrador. Podría tratarse también de varios atacantes activos con diferentes roles dentro del protocolo colaborando entre sí.

Puesto que es posible que el atacante sea una de las partes involucradas en el protocolo, éste podría estar mintiendo o dejar de realizar la parte del protocolo que le corresponde. Este tipo de atacante es llamado **impostor**. Los impostores pasivos siguen el protocolo, pero tratan de obtener más información de la que el protocolo les asigna.

Vulnerabilidad de protocolos de comunicación que emplean criptografía simétrica [33].

¿Cómo es posible comunicar a las partes de manera segura? es de esperar, que las partes encripten sus comunicaciones, sin embargo un protocolo de comunicación que garantice la seguridad entre ellas es más complicado que eso. Veamos que ocurre si Florencia le envía un mensaje cifrado a Francisco:

1. Florencia y Francisco acuerdan entre sí el empleo de un determinado criptosistema.

2. Florencia y Francisco se ponen de acuerdo en la llave que emplearán.

3. Florencia toma el texto en claro y lo encripta empleando el algoritmo y la llave de encripción.

4. Florencia le envía el mensaje cifrado a Francisco.

5. Francisco descifra el mensaje empleando el mismo algoritmo y llave obteniendo el texto en claro.

Pero ¿qué pasa si David, nuestro intruso, se coloca entre Florencia y Francisco escuchando y aprendiendo de lo que ocurre en el protocolo? Supongamos que todo lo que él escucha es la transmisión del mensaje cifrado en el paso cuatro. Seguramente intentará criptoanalizar dicho mensaje. Ya que actualmente existen algoritmos resistentes a ataques que involucren cualquier poder de cómputo que David pueda emplear, será muy difícil para él descifrar este mensaje. Pero David sabe esto, por lo que intentará espiar lo que ocurre en los pasos uno y dos. Si logra hacer esto, conocerá el algoritmo y la llave. De tal forma que cuando un nuevo mensaje encriptado sea enviado por el canal de comunicaciones, todo lo que tiene que hacer es descifrarlo.

Como vimos anteriormente, en un buen criptosistema toda la seguridad es inherente al conocimiento de la llave y no al conocimiento del algoritmo. Es por esto que el manejo de las llaves es tan importante en criptografia. Con un algoritmo simétrico, Florencia y Francisco pueden realizar el paso uno en público, es decir, acordar el criptosistema que van a emplear. Pero el paso dos se debe realizar en secreto, manteniendo la llave oculta durante toda la ejecución del protocolo, tanto tiempo como quieran que el mensaje permanezca cifrado.

Pensemos en un atacante activo y malicioso como Patrick, él puede tratar de hacer varias cosas dañinas: intentar interrumpir las comunicaciones en el paso cuatro, asegurandose de que Florencia no se pueda comunicar con Francisco. También puede interceptar el mensaje de Florencia y sustituirlo por otro. Si Patrick conoce la llave, para lo cual debió interceptar las comunicaciones en el paso dos o romper el criptosistema, puede encriptar su propio mensaje y enviárselo a Francisco en lugar del mensaje original enviado por Florencia. Francisco no tiene forma de saber que dicho mensaje no proviene de Florencia. Si Patrick no conoce la llave, lo más que puede hacer es reemplazar el mensaje que, al ser descifrado por Francisco, no tendrá

ningún sentido. Si Francisco asume que el mensaje viene de Florencia, pensará que la red de comunicaciones empleada o Florencia tienen algún tipo de problema.

Ahora, ¿qué pasa con Florencia? ¿qué puede hacer ella para interrumpir el protocolo? Podría darle una copia de la llave a David, quien podrá leer cualquier cosa que Francisco escriba y envíe a Florencia. Ella podría hacer públicos o emplear contra Francisco los mensajes que él mismo cifre, haciéndolos públicos. Todo esto es indeseable, sin embargo no es un problema del protocolo ya que no hay nada que impida que Florencia le entregue una copia del texto llano a David durante cualquier paso del protocolo. De igual manera Francisco podría hacer las mismas cosas que Florencia. El protocolo presupone que Florencia y Francisco confían plenamente entre sí.

Los criptosistemas simétricos, como el que acabamos de revisar, tienen una serie de problemas:

- Las llaves deben ser distribuidas en secreto. Son más valiosas que los mensajes que encriptan ya que el conocimiento de la llave significa el conocimiento todos los mensajes.
- Si una llave es comprometida (es decir: robada, adquirida bajo extorsión o soborno, adivinada, etc.), entonces David puede descifrar todo el tráfico de mensajes. Pude también suplantar a algunas de las partes produciendo un mensaje falso.
- Si la red de personas que necesitan comunicarse de esta forma no es reducida, podemos pensar en manejar una llave separada para cada par de usuarios de la red. El problema es que el número total de llaves necesarias se incrementa muy rápidamente al aumentar el número de usuarios. Para una red de n usuarios se requerirán $n(n-1)/2$ llaves distintas. Por ejemplo, para un red de cinco usuarios se emplearán 10 llaves diferentes, para una red de 50 usuarios serán necesarias 1,225 llaves diferentes.

Vulnerabilidad de comunicaciones que emplean criptografía de llave pública.

Comencemos retomando la idea de algoritmo simétrico, la cual fue hasta mediados de los setenta, hegemónica en el mundo de la criptografía. ¿qué tan seguro es un algoritmo simétrico? En cierto sentido, tanto como una caja fuerte. En esta analogía, la llave es la combinación, y cualquier persona que la posea puede abrir la caja, colocar un documento adentro y cerrar de

nuevo. Alguien más puede, si conoce la combinación, abrir la caja y sacar el documento. Cualquier persona que no conozca la combinación tendrá necesariamente que aprender a ser un ladrón de cajas fuertes y a forzar las cerraduras, si quiere acceder al contenido de éstas.

En 1976 Whitfield Diffie y Martin Hellman cambiaron el paradigma dominante de la criptografía al sugerir la criptografía de llave pública [17]. Ellos propusieron el uso de dos llaves diferentes, una pública y otra privada. Estas llaves tienen la característica de que es computacionalmente difícil deducir la llave privada a partir de la llave pública. Cualquier persona con la llave pública puede cifrar un mensaje; sin embargo sólo la persona con la llave privada puede descifrarlo.

Veamos ahora un protocolo de este tipo que Florencia puede emplear para enviarle un mensaje a Francisco:

1. Florencia y Francisco acuerdan un criptosistema de llave pública.
2. Francisco le envía a Florencia su llave pública.
3. Florencia encripta el mensaje empleando la llave pública de Francisco y le envía a éste el mensaje cifrado.
4. Francisco descifra el mensaje de Florencia empleando su propia llave privada.

Una de las cosas que se puede observar de este protocolo es que se resuelve el problema del manejo de las llaves que tienen los criptosistemas simétricos en los cuales Florencia y Francisco deben acordar la llave común en secreto o encontrar algún mecanismo seguro para compartir dicha llave. La criptografía de llave pública simplifica el manejo de las llaves: Florencia puede enviarle un mensaje seguro a Francisco. Aún cuando David esté escuchando dicha comunicación, lo más que puede hacer es tomar la llave pública de Francisco, cifrar un mensaje y enviárselo. Pero no puede recuperar el mensaje que Florencia le envió a Francisco, ni tampoco su llave privada. Es decir, lo más que podría intentar es suplantar a Florencia.

Es frecuente que una red o grupo de personas que necesitan comunicarse de manera segura acuerde un criptosistema de llave pública. Cada uno de los usuarios tendrá sus propias llaves tanto pública como privada, las llaves públicas estarán disponibles en una base de datos ubicada en algún lugar. De esta manera el protocolo se simplifica aun más:

1. Florencia obtiene la llave pública de Francisco de la base de datos donde se almacena.

2. Florencia encripta el mensaje empleando la llave pública de Francisco y le envía a éste el mensaje cifrado.

3. Francisco descifra el mensaje de Florencia usando su propia llave privada.

La diferencia entre este protocolo y el anterior consiste en que en el primero de ellos Francisco debe enviarle a Florencia su llave pública, mientras que en el segundo no se involucra hasta que quiera leer el mensaje enviado por Florencia.

3.5. Criptosistemas híbridos

Los primeros algoritmos de llave pública se dieron a conocer al mismo tiempo que el DES (recordemos que es un algoritmo del tipo simétrico) estaba siendo propuesto y discutido como un estándar, lo cuál generó controversia en la comunidad criptográfica [17, 18]. En ese tiempo Whitfield Diffie escribió:

Cita 5.

En el mismo año en que la criptografía de llave pública fue descubierta, la Agencia Nacional de Seguridad de Estados Unidos, la NSA (del inglés: National Security Agency), propuso un sistema criptográfico convencional diseñado por IBM como el estándar federal de encripción de datos (DES). Martin Hellman y yo criticamos esa propuesta sobre la base de que la llave que ésta emplea es muy corta, pero los fabricantes apoyaron dicha propuesta de estandarización y nuestras críticas fueron vistas como un intento de sabotear el proceso de estandarización en favor de nuestro propio trabajo. La criptografía de llave pública fue entonces atacada como si fuera un producto de la competencia en lugar de un resultado reciente de investigación.

W. Diffie, *The First Ten Years of Public Key Cryptography*, *Contemporary Cryptology: The Science of Integrity*, G. Simmons, IEEE Press, 1992.

Todo esto resultó en el hecho de que en la actualidad los algoritmos de llave pública no sustituyen a los algoritmos simétricos. Los primeros no son empleados para cifrar mensajes sino para encriptar las llaves. Las razones de esto son que los algoritmos de llave pública son lentos, aproximadamente mil veces más lentos que los algoritmos simétricos y que son vulnerables a ataques de texto llano elegido. En este último sentido, si tenemos un mensaje cifrado $C = E(M)$, en donde M es un texto en claro de entre n posibles, entonces el criptoanalista, que dispone de la llave pública de encripción, puede cifrar todos los n posibles textos en claro y compararlos con C. Y aunque no pueda determinar la llave privada (de descifrado) sí podrá determinar el mensaje M. Un ataque de éste tipo es particularmente efectivo si hay relativamente pocos mensajes cifrados que sean posibles. Por ejemplo, si el mensaje encriptado M es sólo una coordenada, digamos N 38.000° 51.132'. Es relativamente simple para el criptoanalista cifrar todas las posibilidades, las cuales en principio son: $2 \times 90 \times 1000 \times 360 \times 1000$. Esto es, dos por norte y sur, y tenemos noventa grados con precisión de milésimas para determinar la latitud y trescientos sesenta minutos con precisión de milésimas para la longitud. Incluso si M no está tan claramente definido, un ataque de este tipo podría ser efectivo. En ciertos contextos, el conocer que un determinado texto cifrado no corresponde a un texto claro en particular puede ser información útil. Los criptosistemas simétricos no son vulnerables a estos ataques ya que el criptoanalista no puede realizar pruebas de cifrado con una llave desconocida.

En diferentes implementaciones o programas de comunicaciones seguras la criptografía de llave pública es empleada para asegurar y distribuir las llaves de sesión, luego éstas son utilizadas por los algoritmos simétricos para asegurar el tráfico de mensajes [20]. Esto es a veces llamado un **criptosistema híbrido**. Veamos un protocolo de este tipo:

1. Francisco le envía a Florencia su llave pública.

2. Florencia genera una llave de sesión K, de forma aleatoria y la encripta empleando la llave pública de Francisco, luego se la envía a él, es decir, le envía a Francisco $E_F(K)$.

3. Francisco descifra el mensaje de Florencia empleando su llave privada para recuperar la llave de sesión: $D_F(E_F(K)) = K$.

4. Francisco cifra sus comunicaciones con Florencia empleando la llave de sesión K.

El uso de la criptografia de llave pública resuelve un problema importante en el manejo de llaves, que es el de su distribución. Con la criptografía simétrica, la llave de encriptación de datos está guardada sin hacer nada hasta que es empleada. Si por alguna razón David logra obtenerla, puede desencriptar los mensajes cifrados con ella. En cambio, si se emplea el protocolo anterior, la llave de sesión es creada cuando es necesario cifrar las comunicaciones y es destruida cuando ya no es necesaria. Esto reduce grandemente el riesgo de comprometer la llave de sesión. Desde luego existe el riesgo de comprometer la llave privada, pero es un riesgo más pequeño.

Los rompecabezas de Merkle

Ralph Merkle inventó uno de los primeros mecanismos de criptografía híbrida. En 1974 Merkle se inscribió en un curso de seguridad informática en la Universidad de Berkely, California, impartido por el profesor Lance Hoffman, el tema de su trabajo final de curso trataba el problema de las "comunicaciones seguras en canales inseguros" [34]. El profesor Hoffman no pudo captar la propuesta de Merkle y éste abandonó el curso. Sin embargo continuó trabajando sobre sus ideas a pesar de las dificultades que tenía para hacerse entender [29]. La técnica desarrollada por Merkle se basa en "rompecabezas" que son más fáciles de resolver para el emisor y el receptor que para un intruso o espía. La forma que ideó Merkle para que Florencia pueda enviarle un mensaje encriptado a Francisco sin tener que intercambiar de antemano la llave con el, es la siguiente:

1. Francisco genera 2^{20}, o aproximadamente un millón, de mensajes con la siguiente forma: "Éste es el rompecabezas número X. Ésta es la llave secreta con el número Y", en donde X es un número aleatorio y Y es una llave secreta también aleatoria. Tanto X como Y son diferentes para cada mensaje. Empleando un algoritmo simétrico codifica cada uno de los mensajes con una llave de 20 bits diferente para cada uno y se los envía todos a Florencia.

2. Florencia elige un mensaje al azar y realiza sobre éste un ataque de fuerza bruta para recobrar el texto llano. La cantidad de trabajo que esto requiere es mucha, pero no es imposible de hacer.

3. Florencia cifra su mensaje secreto utilizando algún algoritmo simétrico y la llave que recuperó. Le envía el mensaje cifrado a Francisco junto con el valor X.

4. Francisco conoce llave secreta Y que encriptó el mensaje X, de tal forma que puede descifrar el mensaje.

El intruso David puede quebrar este sistema, pero la cantidad de trabajo que debe realizar para lograrlo es mucho mayor que la que realizan Florencia y Francisco. Para que él pueda recuperar el mensaje en el paso 3 debe realizar un ataque de fuerza bruta en contra de cada uno de los 2^{20} mensajes que Francisco generó en el paso 1. Lo cual es una tarea bastante abrumadora. De manera general, el esfuerzo que David debe realizar para quebrar este sistema es aproximadamente proporcional al cuadrado del esfuerzo que Florencia debe hacer para comunicarse con Francisco por este método. Esta ventaja de n sobre n^2 es pequeña en términos de los estándares criptográficos actuales, pero en algunas circunstancias puede ser suficiente.

El siguiente código genera el archivo MerkleTS.txt que contiene un conjunto de mensajes en claro con la estructura siguiente:

Este es el mensaje No. [2 7 7 8] y su llave es [6 9 6 0 8 8 9 8 8 5 7 7 5 5 2 0 4 8 13]

y el archivo MerkleTC.txt con estos mensajes cifrados empleando un algoritmo de transposición.

Código *Python* 16. Generador de un rompecabezas de Merkle

```
from random import randint
import math
def Cifrar(claro):
    E = claro.rfind(' ')
    columnas=int(claro[E+1:len(claro)-1])
    codificado=''
    claro = claro.replace(' ','') # Se eliminan los espacios
    claro = claro.upper()         # Se pasa a mayusculas
    renglones=int(math.ceil(len(claro)/float(columnas)))
    papel = [['' for x in range(columnas)] for y in range(renglones)]
    claro=claro + '*' * (renglones*columnas - len(claro))
    k=0
    for i in range(renglones):
        for j in range(columnas):
            papel[i][j]=claro[k]
            k+=1
    for j in range(columnas):
        for i in range(renglones):
            codificado=codificado + papel[i][j]
    return codificado
z = 4 # Id del Mensaje
k = 18 # Digitos de la llave
N = 100 # Numero de mensajes
X = [[randint(0,9) for i in range(z)]for i in range(N)]
Y = [[randint(0,9) for i in range(k)]+[randint(3,18)] for i in range(N)]
S = ['Mensaje No. '+str(X[i]).replace(',','') + ' y su llave es '
```

```
    + str(Y[i]).replace(',','') for i in range(N)]
C = [Cifrar(S[i]) for i in range(N)]
archivo=open('MerkleTS.txt','w')
for i,s in enumerate(S):
    archivo.write(str(i)+','+s+'\n')
archivo.close()
archivo=open('MerkleTC.txt','w')
for c in C:
    archivo.write(c+'\n')
archivo.close()
print('Se generaron%d mensajes con la estructura:'% N)
print(S[0])
print('Se generaron los mensajes simples en MerkleTS.txt')
print('Se generaron los mensajes cifrados en MerkleTC.txt')
```

Este programa lee el archivo con los mensajes cifrados, elige uno de forma aleatoria y le realiza un ataque de fuerza bruta, recuperando la llave.

Código *Python* 17. Ataque de fuerza bruta a Merke

```
from random import randint
import math
def Descifrar(codificado,columnas):
    claro=''
    renglones=int(math.ceil(len(codificado)/float(columnas)))
    papel = [['' for x in range(renglones)] for y in range(columnas)]
    codificado=codificado + '*' * (renglones*columnas - len(codificado))
    k=0
    for i in range(columnas):
        for j in range(renglones):
            papel[i][j]=codificado[k]
            k+=1
    for i in range(renglones):
        for j in range(columnas):
            claro=claro + papel[j][i]
    return claro
archivo=open('MerkleTC.txt','r')
C = [c.strip() for c in list(open('MerkleTC.txt','r'))]
N = len(C)
n = randint(0,N-1)
print('Se recibieron%d mensajes'% N)
print('Descifrando por FB, el numero%d:'% n)
print(C[0])
z=0; D=''
while D.find('MENSAJE')==-1:
    z+=1; k = randint(2,20)
    D = Descifrar(C[n],k)
    D = D.replace('*','')
print('El mensaje es:', D)
print('Se realizaron%d iteraciones.'% z)
```

3.6. Funciones de una vía

La noción de función de una vía es central en la criptografia de llave pública. Aunque no es parte de los protocolos en sí mismos, las **funciones de una vía** son bloques fundamentales en la mayoría de los protocolos que estudiaremos [28, 29].

De manera informal podemos decir que éstas son funciones matemáticas fáciles de calcular, pero considerablemente difíciles de "revertir". Esto es: dado un valor de x es fácil calcular $f(x)$, pero si se tiene solamente el valor que toma $f(x)$ es muy difícil encontrar el valor de x. Una analogía útil es la siguiente: es fácil romper un plato de cerámica en pedazos, pero es considerablemente más difícil juntar todos los pedazos, hasta los más pequeños, y reensamblar el plato. Este tipo de funciones constituye un campo de trabajo en las matemáticas contemporáneas.

Un tipo particular de estas funciones es empleado en la criptografía de llave pública: las funciones de una vía con **puerta trasera**. Con ellas es fácil calcular en una dirección y muy difícil calcular en la dirección contraria. Pero, si uno conoce el secreto de la puerta trasera, resulta sencillo revertir la función. La idea se puede representar así: Es simple calcular $f(x)$ dado el valor de x. Es muy difícil calcular x si sólo se conoce $f(x)$, a menos de que se conozca cierta información secreta. Una analogía que puede servir para completar la idea es la siguiente: Es simple desarmar un reloj mecánico en los miles de piezas que lo componen, pero si alguien nos entrega una bolsa con ese montón de piezas y nos pide que armemos el reloj y lo pongamos a andar, las cosas se complican. Sin embargo, si se nos entregan además las instrucciones detalladas (secretas) de ensamblado, resulta relativamente simple ensamblar de nuevo el reloj.

Un algoritmo o función *hash* es una función especial de una vía que toma como entrada una cadena de longitud variable, llamada **preimagen**, y la convierte en una cadena de longitud fija, generalmente más pequeña, llamada **valor hash** o simplemente hash.

El siguiente código implementa una función hash que toma una cadena de entrada, de cualquier longitud, y regresa una cadena numérica de 5 caracteres.

Código *Python* 18. Generador de una función Hash

```
def hash01(Cadena):
    p=0; s=0; M=50000
    for i,m in enumerate(Cadena):
        p=p+ord(m)
        z=256**i
        s=s+z*ord(m)
    St=str(s%M)
    St=(len(str(M))-len(St))*str(p%10)+St
    return(St)

Mensaje=input('Dame la cadena de entrada ')
print('La salida de la funcion Hash1 es:', hash01(Mensaje))
```

Otro ejemplo sería una función que toma una preimagen determinada, como una cadena de bytes de cualquier longitud, y entrega como resultado un solo byte obtenido de realizar la operación XOR sobre todos los bytes de la cadena de entrada. En primera instancia, podría pensarse que es una función hash de una vía ya que coincide con parte de la descripción dada anteriormente. Sin embargo, no es una buena función hash, ya que tiene "colisiones", es decir, dado un byte como valor específico de salida es simple generar una cadena de bytes cuyo valor XOR de salida sea el mismo. Si una función de este tipo tiene la propiedad de que es difícil generar dos preimágenes con el mismo valor hash, se dice que está **libre de colisiones**.

En síntesis podemos decir que las funciones hash de una vía tienen las siguientes propiedades:

1. Ser de una sola vía: es fácil tomar un mensaje y calcular su valor hash, pero es muy difícil, o imposible, tomar el valor hash y recuperar el mensaje original.

2. Estar libres de colisiones: es muy difícil encontrar dos mensajes que generen el mismo valor hash.

En criptografia quebrar una función hash significa mostrar que cada una o ambas de estas propiedades no son ciertas.

Las funciones hash reciben varios nombres que de alguna manera reflejan el uso que se les da: funciones de compresión, funciones de contracción, huella digital, control de integridad del mensaje o MIC, suma de control criptográfico, etc. Lo importante, en términos de la utilidad que tienen, es que se puede generar una huella de la preimagen, que es a veces llamada *clave*. Es decir, la función origina un valor que nos permite distinguir si una preimagen cualquiera es la preimagen que generó el valor hash o clave,

permitiéndonos verificar si ésta es la preimagen real. Ya que las funciones hash son típicamente de la forma "muchos a uno" (no inyectivas y sobreyectivas), no es posible emplearlas para determinar con certeza si dos cadenas o preimágenes son iguales; a pesar de esto, proporcionan una certeza razonable. Una de las propiedades fundamentales del **hashing** es que si dos resultados de una misma función son diferentes, entonces las dos entradas que generaron dichos resultados también lo son. Como ya hemos dicho, es posible que existan valores hash resultantes iguales para objetos diferentes, ya que el rango de posibles claves es mucho menor que el de posibles objetos a resumir.

Como ya mencionamos, una de las utilidades de las funciones hash es que proporcionan un mecanismo de **huellas digitales** para archivos. Supongamos la siguiente situación: Estás interesado en verificar si alguien más, digamos Arturo, tiene un archivo determinado que tú también posees. Debido a lo sensible de la información que éste contiene, Arturo no está dispuesto a mostrarte el archivo a pesar de que tú afirmas también tenerlo. Una forma de verificar si Arturo tiene el archivo secreto es pedirle que te envíe el valor hash. Si Arturo envía el valor correcto es prácticamente seguro que tenga el mismo archivo.

Otra de las utilidades de las funciones hash es en los códigos de autenticación de mensajes o MAC (por sus siglas en inglés: Message Authentication Codes). Éstos consisten en una función hash con la adición de una llave secreta. El valor de hash es entonces función de la preimagen y la llave, funciona como la huella digital de un archivo, excepto porque solamente alguien con la llave puede verificar el valor hash.

Capítulo 4

Firmas digitales

Las firmas hechas a mano has sido utilizadas a lo largo de la historia como pruebas de autoría o de acuerdo con el contenido del documento firmado. Pero, ¿qué es lo que hace que las firmas sean útiles y atractivas de emplear? Veamos algunas de sus cualidades teóricas:

1. Son una prueba de autenticidad; la firma tiene la capacidad de convencer al destinatario del documento de que ésta se hizo de forma deliberada por su autor.

2. Son infalsificables; la firma representa una prueba de que su autor, y nadie más que él, deliberadamente firmó el documento.

3. No son reutilizables; La firma es parte misma del documento, una persona con pocos escrúpulos no puede trasladar la firma a un documento diferente.

4. El documento firmado es inalterable. Después de que un documento ha sido firmado, este no puede ser alterado.

5. La firma no puede ser repudiada. Tanto la firma como el documento son objetos físicos; el autor no podrá negar la autoría de su firma sobre el documento.

Sabemos que éstas son sólo cualidades teóricas ya que ninguna de las cosas mencionadas es completamente cierta. Las firmas pueden ser falsificadas, pueden copiarse de un documento a otro, los documentos pueden modificarse después de haber sido firmados, etc. Sin embargo, y a pesar de estos problemas las firmas se siguen empleando, entre otras cosas porque hacer trampa involucra cierto riesgo de detección.

¿Qué ocurre con las firmas en el mundo digital? Aparecen ciertos problemas: los archivos de computadoras son muy fáciles de copiar. Aun cuando la firma de una persona sea difícil de falsificar, es fácil obtener una imagen, de tal forma que se pueda copiar y pegar de un documento a otro; la presencia de dicha firma pierde sentido y no significa nada. Además, los archivos de computadoras son muy simples de modificar después de que han sido firmados, incluso sin dejar evidencia de dicha modificación [35].

4.1. Firma de documentos empleando un criptosistema simétrico y un árbitro

Pensemos en una situación en la cual Florencia necesita firmar un mensaje digital y enviárselo a Francisco. Si ella está dispuesta a confiar en un árbitro, como lo sería nuestro personaje Ángel, y a emplear además un criptosistema simétrico la tarea es posible utilizando un protocolo.

En el juego de roles, Ángel es un árbitro completamente confiable, puede comunicarse con Florencia y Francisco, o con cualquier otra persona interesada en firmar digitalmente un documento. También, comparte una llave secreta K_F con Florencia, y una llave secreta diferente con Francisco K_{Fr}. Ambas llaves han sido acordadas antes de que el protocolo se lleve a cabo, dichas llaves pueden ser reutilizadas. Consideremos el siguiente procedimiento:

1. Florencia encripta el mensaje para Francisco utilizando K_F y se lo envía a Ángel.

2. Ángel descifra el mensaje empleando K_F

3. Ángel toma el mensaje en claro y un documento en donde asegura haber recibido dicho mensaje de Florencia, encripta ambos con la llave que comparte con Francisco K_{Fr}. y se los envía a Francisco.

4. Francisco desencripta el paquete con su llave K_{Fr}. Puede ahora leer tanto el mensaje que Florencia le envió, como el certificado de Ángel asegurando que el mensaje lo recibió de Florencia.

En este protocolo, la pregunta interesante es: ¿cómo sabe Ángel que el mensaje realmente proviene de Florencia y no de algún impostor? En realidad lo que Ángel hace es inferir, por la encripción, que el mensaje proviene de Florencia. Es decir, puesto que solamente Florencia y él comparten la

llave secreta K_F, por lo que se espera que solamente Florencia pueda cifrar un mensaje empleando dicha llave. Este procedimiento tiene todas las características que ya mencionamos respecto a las firmas autógrafas:

La firma tiene autenticidad. Esto es, Ángel es un árbitro confiable y "sabe" que el mensaje proviene de Florencia. El certificado de Ángel sirve como una prueba de esto para Francisco.

La firma es infalsificable: solamente Florencia y Ángel conocen la llave K_F, por lo que únicamente Florencia le puede enviar un mensaje a Ángel encriptado con esa llave. Si alguien trata de suplantar a Florencia, Ángel se dará cuenta de ello y no certificará la autenticidad del mensaje.

La firma no es reutilizable. En el caso de que Francisco intentara emplear la certificación de Ángel en otro mensaje, lo que pasaría es que el árbitro de esa comunicación (que podría ser el mismo Ángel o algún otro con la misma confianza y acceso a la misma información) le pediría a Francisco que enviara tanto dicho mensaje *fraudulento* como el mensaje cifrado de Florencia. De esta manera el árbitro podrá cifrar con K_F y darse cuenta de que no concuerda con el mensaje fraudulento de Francisco ya que este último no puede producir un mensaje cifrado que concuerde ya que no conoce la llave de Florencia K_F.

El documento firmado es inalterable. Si Francisco tratara de alterar el documento después de haberlo recibido, el árbitro podría probar que éste ha sido alterado de la misma manera como se explicó en el párrafo anterior.

La firma no puede ser repudiada. Si Florencia quisiera negar a Francisco que ella fue quien envió el mensaje, la certificación de Ángel probaría lo contrario.

En el caso de que Francisco quisiera mostrarle a un tercero, digamos a Clemente, el documento firmado por Florencia tendría que recurrir de nuevo a un árbitro o, de lo contrario revelarle su llave secreta K_{Fr} a dicho tercero. Si decide no revelar su llave secreta entonces el protocolo arbitrado sería el siguiente:

1. Francisco toma el mensaje de Florencia que desea compartir, así como el certificado de Ángel que prueba que dicho mensaje proviene de Florencia. Los encripta con su llave secreta K_{Fr} y se los envía de nuevo a Ángel.

2. Ángel desencripta el paquete con la llave que comparte con Francisco K_{Fr} .

3. Ángel verifica en su base de datos y confirma si el mensaje original proviene de Florencia.

4. Ángel encripta el paquete con la llave secreta que comparte con Clemente K_C y se lo envía a éste.

5. Clemente descifra el paquete con su llave K_C, con esto puede leer tanto el mensaje original, como el certificado de Ángel confirmando que Florencia lo envió.

Un protocolo de este tipo funciona pero tiene el inconveniente de consumir mucho tiempo del árbitro, el cual deberá pasar horas cifrando y descifrando mensajes al actuar como intermediario entre cualquier par de personas que necesiten enviarse documentos firmados entre sí. Deberá además mantener la base de datos de los mensajes, aunque esto podría evitarse si le envía a los receptores las copias de los mensajes cifrados del emisor. De cualquier manera, el árbitro se convertirá en el cuello de botella del sistema de comunicaciones. Una dificultad importante resulta en que es difícil crear y mantener a un árbitro como Ángel: alguien en el cuál todos los usuarios de la red de comunicaciones confíen. Puesto que deberá ser infalible. Deberá ser también completamente seguro; si su base de datos de llaves secretas compartidas es vulnerada de alguna manera, dichas firmas se vuelven completamente inútiles. Aunque teóricamente este protocolo puede ser útil, en la práctica no funciona muy bien.

Firma de documentos empleando criptografía de llave pública.

Existen varios algoritmos de llave pública que pueden ser empleados para hacer firmas digitales. En algunos de estos algoritmos, como el RSA, tanto la llave pública como la privada pueden ser empleadas para encriptar mensajes; sin embargo, si se emplea la llave privada, lo que se obtiene es una firma digital segura, ya que nadie más que el autor conoce su llave privada. En otros casos, existe un procedimiento separado para las firmas digitales, el cual es distinto del que se emplea para la encripción.

La idea de firmar documentos empleando criptografia de llave pública fue originalmente propuesta por Diffie y Hellman y el protocolo básico es el siguiente:

1. Florencia cifra el mensaje con su llave privada, al hacerlo también está firmando el documento.

2. Florencia le envía a Francisco el documento cifrado y "firmado".

3. Francisco descifra el documento utilizando la llave pública de Florencia, verificando así la firma sobre dicho documento.

Este protocolo mejora al anterior ya que no es necesaria la presencia del árbitro para realizar o verificar las firmas, aunque sigue siendo necesario para certificar la llave pública de Florencia. Las partes no necesitan del árbitro para resolver posibles disputas ya que si Francisco no puede realizar el paso 3 sabe entonces que la firma que lleva el documento es inválida. Además el protocolo satisface los requisitos que buscamos en una firma, ya que tiene autenticidad, puesto que cuando Francisco descifra el mensaje con la llave pública de Florencia inmediatamente sabe que Florencia lo firmó al cifrarlo con su llave privada. Sabemos que la firma es infalsificable, ya que sólo Florencia conoce su llave privada. La firma no es reutilizable, ya que "actúa" sobre un documento específico y no puede ser transferida a otro, es propia del documento que cifra. Asimismo, el documento firmado es inalterable, ya que si es modificado de alguna manera no podrá ser verificado con la llave pública de Florencia. Tampoco puede ser rechazada, ya que sólo Florencia conoce su llave privada.

4.2. Firma de documentos y estampas temporales

Pensemos en una situación en la cual Francisco tratará de engañar a Florencia empleando un documento firmado por ella. En este caso, el documento es un cheque.

Digamos que Florencia le envía a Francisco un **cheque digital** firmado por un valor de $100,000 pesos. Francisco deposita el cheque virtual en el banco, el cuál verifica la firma y mueve el dinero de una cuenta a otra. Pero Francisco, maliciosamente, guarda una copia del cheque digital y unos días después lo "deposita" de nuevo en el banco, tal vez en un banco diferente. El banco verifica entonces la firma y mueve el dinero de una cuenta a otra, de tal forma que si Florencia no revisa constantemente su estado de cuenta podría ser desfalcada.

Debido a situaciones como ésta, es que las firmas digitales frecuentemente llevan una estampa de tiempo [36]: Tanto la fecha como la hora de la firma son agregadas al mensaje, luego el paquete completo es firmado. De esta manera, el banco guarda la estampa de tiempo en su base de datos. Si Francisco intenta usar la copia del cheque digital una segunda vez, el banco verifica la estampa temporal en su base de datos y se da cuenta de que el cheque ya ha sido empleado, negando la transferencia y tomando algún tipo de medidas en contra del fraude.

4.3. Firma de documentos empleando criptografía de llave pública y funciones de una sola vía

En la mayoría de las aplicaciones prácticas, los algoritmos de llave pública resultan ineficientes en términos del tiempo que emplean para firmar documentos extensos. Para ahorrar tiempo, algunos protocolos de firma digital utilizan funciones de una vía. Esto es, en lugar de firmar el documento completo solamente se firma una parte de éste. En un protocolo así tanto la función de una vía como el algoritmo de firma digital son acordados de antemano. El protocolo es entonces:

1. Florencia produce el valor hash del documento en cuestión.
2. Florencia cifra el valor hash del documento empleando su llave privada, firmando así el hash.
3. Florencia le envía a Francisco el documento en claro y el valor hash firmado.
4. Francisco genera el valor hash del documento que Florencia le envió. Luego, empleando el algoritmo de firma digital, descifra el valor hash firmado por Florencia empleando la llave pública que ella empleó. Si este valor hash enviado por Florencia y el generado por Francisco concuerdan, entonces se concluye que la firma es válida.

Un protocolo de este tipo mejora considerablemente en rapidez. Además de tener otros beneficios como que la firma y el documento estén separados y disminuyan los requisitos de almacenamiento del receptor. Un sistema de archivos puede emplear estos protocolos para verificar la existencia de documentos sin almacenarlos necesariamente, ya que la base de datos puede así almacenar únicamente el valor hash de dichos archivos, de tal manera que los usuarios envían estos valores, la base de datos genera una estampa temporal de éstos y luego los almacena. En caso de presentarse algún desacuerdo o duda sobre quién y cuándo creó un cierto documento, la base de datos lo puede resolver al comparar los valores hash almacenados.

Existen varios algoritmos de firma digital. Todos ellos de llave pública, lo que implica información secreta para firmar los documentos e información pública para verificar las firmas. Algunas veces el proceso de firmar es llamado **encripción con llave privada**, el proceso de verificación es, a su vez, llamado **desencripción con llave pública**. Esto es correcto para el caso del algoritmo RSA, pero ha generado cierta confusión ya que los diferentes

algoritmos tienen formas distintas de realizarse y muchos pueden emplearse para firmas digitales, pero no para el cifrado. Para evitar la confusión, nos referiremos a los procesos de firma y verificación sin dar más detalles del algoritmo empleado. De tal manera que al proceso de firmar un mensaje empleando una llave privada K, lo escribiremos como $S_K(M)$. Al proceso de verificar la firma utilizando la llave pública lo escribiremos como: $V_K(M)$. A la cadena de bits asociada a un documento que ha sido firmado le llamaremos firma digital o simplemente firma. El protocolo completo consistente en que el receptor de un mensaje ha sido convencido de la identidad del emisor y la integridad del mensaje es llamado **autenticación**.

4.4. Múltiples firmas en un documento

Ahora abordaremos un problema frecuente asociado a las firmas: ¿Qué ocurre si más de una persona debe firmar un documento? En otras palabras, ¿Cómo pueden Florencia y Francisco firmar el mismo documento digital? Existen varias formas de hacerlo, una de ellas es que tanto Florencia como Francisco firmen copias separadas del documento, con el inconveniente de que el mensaje resultante será del doble del tamaño del documento original. Otra forma de resolverlo es que Florencia firme primero el documento y luego Francisco firme la firma de Florencia sobre el documento. Ésto, aunque funciona, hace que sea imposible verificar la firma de Florencia sin verificar la de Francisco. Una tercera solución al problema es empleando funciones hash, el procedimiento es el siguiente:

1. Florencia firma el hash del documento.
2. Francisco firma el hash del documento.
3. Francisco envía el hash firmado a Florencia.
4. Florencia envía el documento, el hash firmado y el de Francisco, al árbitro Ángel.
5. Ángel verifica tanto la firma de Florencia como la de Francisco en su base de datos.

Florencia y Francisco pueden realizar los pasos 1 y 2, de forma independiente. En el paso 5 el árbitro puede verificar la firma cualquiera de los dos sin necesidad de verificar la otra.

Rechazo de las firmas digitales

Florencia podría intentar hacer trampa con las firmas digitales de la siguiente manera: firmando un documento y después negarlo. ¿Cómo?, primero firma el documento de manera normal. Luego, de forma anónima, publica su llave privada. Esto le permite asegurar que su firma ha sido comprometida, que ella no tiene nada que ver con eso y que otros la están usando pretendiendo ser ella. De esta manera puede rechazar el haber firmado el documento en cuestión, además de otros documentos firmados utilizando su llave privada. Esto es llamado **repudio**.

Las estampas temporales pueden limitar los efectos de este tipo de trampa, pero Florencia siempre puede argumentar que su llave fue comprometida con anterioridad. Si sincroniza las cosas de manera adecuada, ella puede firmar un documento y luego aseverar exitosamente que no lo hizo. A pesar de que no es posible evitar este tipo de abuso, se pueden tomar una serie de medidas para garantizar que las firmas ya realizadas no se invaliden por futuras acciones tomadas con el propósito de hacer trampa. Una solución es que el receptor de un documento firmado emita una estampa temporal como en el siguiente protocolo:

1. Florencia firma el mensaje.

2. Florencia genera un encabezado para el mensaje conteniendo alguna información que lo identifique. Luego concatena el encabezado con el mensaje firmado, firma el paquete y se lo envía a Ángel.

3. Ángel verifica la firma externa del paquete y confirma la información del encabezado. Además, pone una estampa temporal al mensaje firmado por Florencia y a la información del encabezado. Luego, firma este paquete y se los envía a Florencia y Francisco.

4. Francisco verifica la firma de Ángel, la información del encabezado y la firma de Florencia.

5. Florencia verifica el mensaje que Ángel le envió a Francisco. En el caso de que ella no originara dicho mensaje lo comunica rápidamente.

Una de las primeras aplicaciones para el empleo de firmas digitales fue en la verificación de pruebas nucleares: La antigua Unión Soviética y los E.U.A. permitieron uno al otro colocar sismógrafos para monitorear las pruebas nucleares que se realizaran. Uno de los problemas era que cada uno necesitaba

asegurarse de que el otro no interfiriera o modificara la información proveniente de los sismógrafos colocados en el otro país. Simultáneamente, la nación huésped necesitaba estar segura de que los monitores enviaban únicamente la información acordada y necesaria para dicha tarea. Las técnicas convencionales de autenticación podían resolver el primer problema, pero solamente las firmas digitales podían resolver ambos. De tal manera que la nación huésped podía leer, pero no alterar la información de los sismógrafos extranjeros y la nación que realizaba el monitoreo en un territorio que no era el suyo, garantizaba que la información no era alterada.

Firmas digitales con encriptación

Al combinar la criptografia de llave pública con las firmas digitales, es posible desarrollar un protocolo que combine la seguridad de la encripción con la autenticidad de las firmas digitales:

1. Florencia firma el mensaje con su llave privada: $F_E(M)$
2. Florencia cifra el mensaje firmado con la llave pública de Francisco y se lo envía: $C_F(F_E(M))$
3. Francisco descifra el mensaje con su llave privada: $D_F(C_F(F_E(M))) = F_E(M)$
4. Francisco verifica y recupera el mensaje con la llave pública de Florencia: $V_E(F_E(M)) = M$

Observemos que en este protocolo la firma va implícita en el cifrado, es decir, se firma al cifrar.

El firmar antes de cifrar parece algo natural; cuando alguien escribe una carta, primero la firma y después la coloca en el sobre. Si se colocara la carta sin firmar en el sobre y luego se firmara el sobre, cabría la duda de si la carta no fue reemplazada al mostrársela a un tercero, ya que mostraríamos la carta sin firmar y el sobre firmado. De igual manera con la correspondencia electrónica el hecho de firmar antes de cifrar es una práctica prudente.

No es recomendable utilizar el mismo par de llaves pública-privada, para la encripción y la firma. En el ejemplo tratado en el protocolo anterior, el cifrado equivale a la firma. Es mejor emplear dos juegos distintos de pares de llaves, un juego para el cifrado y otro para las firmas. Esta separación tiene ciertas ventajas ya que la pérdida de un juego no afecta al otro. En cualquier caso, se deben emplear estampas temporales en este tipo de protocolos para prevenir el reciclado de mensajes.

El ataque del reenvío de mensajes

Pensemos en un protocolo en el cual se firma al cifrar, como el presentado en la sección anterior, con la característica adicional de que es necesario confirmar la recepción del mensaje. Cada vez que se recibe un mensaje se envía la confirmación de su recepción:

1. Florencia firma el mensaje con su llave privada, luego lo cifra con la llave pública de Francisco y se lo envía a éste: $C_{Fc}(F_{Fl}(M))$

2. Francisco descifra el mensaje empleando su llave privada y verificando la firma con la llave pública de Florencia, verificando de esta manera que Florencia firmó el mensaje y descifrándolo a su vez: $V_{Fl}(D_{Fr}(C_{Fr}(F_{Fl}(M)))) = M$

3. Francisco firma el mensaje con su llave privada, luego lo cifra con la llave pública de Florencia y lo envía de regreso a ésta: $C_{Fl}(F_{Fr}(M))$.

4. Florencia descifra el mensaje con su llave privada y verifica la firma con la llave pública de Francisco. Si el mensaje que resulta es el mismo que ella envió, entonces sabrá que Francisco recibió su mensaje.

En el caso de que se empleé el mismo algoritmo para la encripción y para la verificación de la firma digital, la operación de firma digital es la inversa de la operación de cifrado, es decir:

$$V_X = C_X \qquad \text{y} \qquad F_X = D_X,$$

en donde C corresponde al cifrado, V al descifrado, F a la firma y D al descifrado.

Veamos ahora cómo es posible atacar este tipo de protocolos. Pensemos en que Patrick, el atacante activo y malicioso, es un usuario legítimo del sistema debido a lo cual posee su propio par de llaves pública-privada. ¿Qué debe hacer Patrick para leer las comunicaciones de Francisco? Primero deberá grabar el mensaje que Florencia le envía a Francisco en el paso 1 del protocolo. Un tiempo después, le envía a Francisco ese mismo mensaje teniendo ahora como emisor al propio Patrick. Francisco pensará que éste es un mensaje legítimo de Patrick ya que no tiene forma de saber que no lo es, de tal manera que descifra el mensaje con su llave privada y luego trata de verificar la firma de Patrick descifrando el mensaje con la llave pública de éste. El mensaje que resulta es un texto ininteligible producto de los procesos:

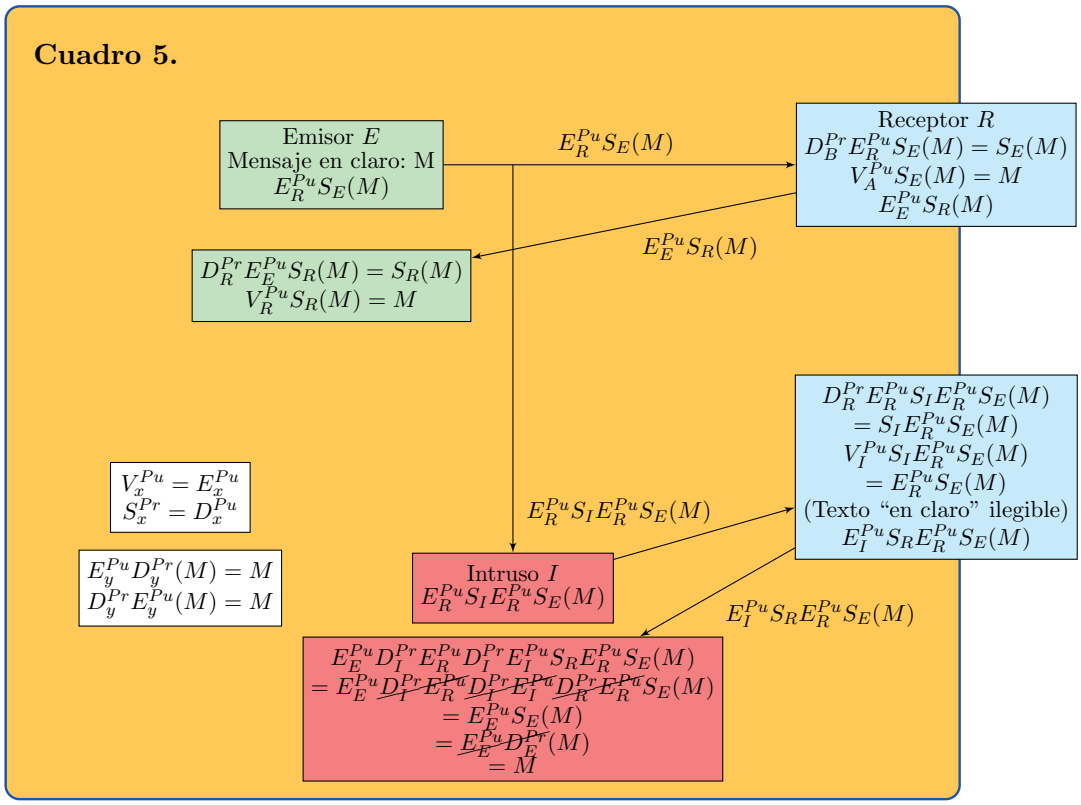
$$C_P(D_{Fr}(C_{Fr}(D_{Fl}(M)))) = C_P(D_{Fl}(M))$$

Continuando con el protocolo, Francisco le envía a Patrick el siguiente recibo del mensaje:

$$C_P(D_{Fr}(C_P(D_{Fl}(M))))$$

Ahora lo que Patrick tiene que hacer es descifrar este mensaje con su propia llave privada, encriptarlo con la llave pública de Francisco y de nuevo descifrarlo con su propia llave privada. Después cifrarlo de nuevo pero esta vez con la llave pública de Florencia y asunto concluido: Patrick tiene el mensaje en claro M (recordemos que $V_X = C_X$ y $F_X = D_X$). A continuación mostramos un diagrama que ilustra este procedimiento:

Cuadro 5.



No es descabellado pensar que Francisco le enviará automáticamente el recibo a Patrick, puesto que el protocolo podría estar en el *software* de comunicaciones, de tal manera que los recibos se enviarán automáticamente. La debilidad de este tipo de ataque se genera por la buena voluntad de enviar el recibo del texto ininteligible; si Francisco revisara la comprensibilidad del mensaje antes de enviar el recibo, podría evitar esta falla en la seguridad. Existen varias mejoras a este tipo de ataque, las cuales le permiten a Patrick enviarle a Francisco un mensaje diferente al que éste obtuvo de Florencia. La conclusión de todo esto es que no es conveniente firmar mensajes provenientes de otras personas sin antes verificar su contenido.

Este ataque funciona debido a que la operación de encripción es la misma que la de firma y la operación de descifrado es la misma que la de verificación. Un protocolo seguro debe emplear, al menos, una operación ligeramente diferente para la encripción y la firma digital. El emplear juegos de llaves diferentes para cada operación o el utilizar algoritmos diferentes resuelve este problema, como se hace en el protocolo siguiente.

1. Florencia firma el mensaje.
2. Florencia cifra el mensaje y lo firma con la llave pública de Francisco (empleando un algoritmo de cifrado diferente al empleado para realizar la firma), y se lo envía a él.
3. Francisco descifra el mensaje con su llave privada.
4. Francisco verifica la firma de Florencia.

También puede mejorarse la seguridad mediante el empleo de estampas temporales, ya que esto tiene como consecuencia que el mensaje de llegada sea diferente al de salida.

Ataques a la criptografía de llave pública

En los protocolos de criptografía de llave pública que hemos revisado, pasamos por alto el mecanismo por el cual Florencia obtiene la llave pública de Francisco. Esto es importante, por lo que observaremos su funcionamiento.

La manera mas sencilla de obtener la llave pública de alguien es buscarla en una base de datos "segura" colocada en algún lugar. Dicha base de datos debe ser accesible a todos los usuarios del protocolo, de tal forma que cualquier usuario pueda obtener la llave pública de otro. La base de datos debe estar protegida de tal modo que sólo un árbitro confiable pueda modificarla y nadie más pueda sustituir las llaves públicas ahí almacenadas.

Supongamos que las llaves públicas están almacenadas en una base de datos segura. Aun así, un intruso malicioso como Patrick podría sustituir las llaves durante la transmisión. Para prevenir esto, el árbitro Ángel puede firmar cada llave pública con su llave privada, generando así una llave certificada por una autoridad, en este caso dicha autoridad es el **KDC** (del inglés **Key Distribution Center**). En la práctica, el KDC firma un mensaje compuesto que consiste en el nombre del usuario, su llave pública y cualquier otra información importante que identifique al usuario. Este mensaje es almacenado en la base de datos del KDC. En el caso de que Florencia obtenga de la base de datos la llave pública de Francisco, verificará la firma del KDC para asegurarse de la validez de dicha llave. Sin embargo, esto no hace imposible el ataque de un intruso, aunque sí le dificultan las cosas. La llave pública del KDC debe a su vez estar almacenada en algún lugar; el intruso podría sustituir esta llave, luego corromper la base de datos general y sustituir el conjunto de llaves válidas almacenadas por otras que, desde luego, estarían firmadas con su llave privada como si fuera el KDC.

Capítulo 5

Protocolos de intercambio de llaves y autenticación

Una técnica criptográfica común consiste en cifrar cada comunicación individual con una llave empleada solamente para una sesión de comunicaciones específica. Dicha llave es llamada **llave de sesión** y existe únicamente por el tiempo que dura la sesión de comunicaciones. A continuación estudiaremos algunos protocolos para distribuir las llaves de sesión.

5.1. Un protocolo de intercambio de llaves de sesión empleando criptografía simétrica

Este protocolo supone que tanto Florencia como Francisco comparten cada uno una llave secreta (distinta a la llave de sesión) con el KDC. En este caso el árbitro confiable será Ángel. Antes de iniciar el protocolo, cada usuario debe conocer su propia llave secreta. La distribución de estas llaves, una vez generadas y asignadas, supone un problema importante de comunicación segura que no se detalla. El protocolo es el siguiente:

1. Florencia se comunica con Ángel y le solicita una llave de sesión para comunicarse con Francisco.

2. Ángel genera aleatoriamente una llave de sesión, luego encripta dos copias de ésta: una la cifra con la llave de Florencia y la otra con la llave de Francisco. Después le envía ambas copias a Florencia.

3. Florencia descifra su copia de la llave de sesión.

4. Florencia le envía a Francisco la copia que le corresponde de la llave de sesión.

5. Francisco descifra su copia de la llave de sesión.

6. Tanto Florencia como Francisco emplean la llave de sesión para comunicarse.

Este protocolo confía totalmente en la seguridad que aporta el árbitro Ángel (KDC), el cual es generalmente un programa confiable más que un individuo. Si el intruso Patrick logra de alguna manera corromper al árbitro, toda la red de comunicaciones arbitrada queda comprometida. De esta forma, Patrick dispondrá de todas las llaves secretas que el árbitro comparte con cada uno de los usuarios, ganando así acceso a todo el tráfico de comunicaciones. Todo lo que el intruso debe hacer a partir de esto es espiar las líneas de comunicación para escuchar el tráfico de mensajes cifrados.

Otro de los problemas asociados a este protocolo es que el árbitro es potencialmente un cuello de botella ya que se debe involucrar en cada uno de los intercambios de las llaves de sesión. Si por alguna razón el árbitro falla el sistema de comunicaciones se viene abajo.

5.2. Un protocolo de intercambio de llaves de sesión empleando criptografía de llave pública

En este protocolo Florencia y Francisco emplearán criptografía de llave pública para acordar una llave de sesión y con ésta poder cifrar la comunicación. En algunos casos puede que las llaves públicas firmadas por el KDC estén disponibles en una base de datos, lo cual facilita el intercambio de llaves permitiendo que Florencia le envíe un mensaje seguro a Francisco, aun si él nunca ha oído hablar de ella. El protocolo se puede realizar siguiendo este procedimiento:

1. Florencia obtiene la llave pública de Francisco del KDC.

2. Florencia genera aleatoriamente una llave de sesión, luego la cifra empleando la llave pública de Francisco y se la envía a éste.

3. Francisco descifra el mensaje de Florencia empleando su llave privada.

4. Ambos encriptan su comunicación empleando la misma llave de sesión.

En el caso de tener a un espía o un intruso que escucha secretamente la conversación en este protocolo, llamémosle David, éste intentará vulnerar la seguridad de las comunicaciones rompiendo el algoritmo de llave pública con un ataque de texto cifrado. Pero, en el caso de un intruso malicioso como Patrick que es un usuario legítimo del sistema de comunicaciones, las cosas se dificultan ya que éste tiene más capacidades y recursos que David. Este intruso no se limitará a escuchar pasivamente los mensajes entre Florencia y Francisco, sino que intentará modificarlos, borrarlos o generar mensajes totalmente nuevos con la intención de vulnerar las comunicaciones. Podrá también suplantar a cualquier usuario de la red. Supongamos que de alguna manera se coloca en medio del canal de comunicaciones. Un ataque de este tipo es conocido como **hombre en medio del camino** y toma la siguiente forma:

1. Florencia le envía a Francisco su llave pública. Patrick intercepta esta llave, la sustituye por la suya y se la envía a Francisco.

2. Francisco le envía a Florencia su llave pública. Patrick intercepta esta llave, la sustituye por la suya y se la envía a Florencia.

3. Cuando Florencia le envía un mensaje a Francisco cifrándolo, con lo que ella cree que es la llave pública legítima de éste, Patrick lo intercepta. Puesto que el mensaje está cifrado con la llave pública que Patrick sustituyó, éste puede descifrarlo con su llave privada, luego reencriptarlo con la llave pública de Francisco y enviárselo a este último.

4. Cuando Francisco le envía un mensaje a Florencia cifrándolo, con lo que cree que es la llave pública de Florencia, Patrick lo intercepta. Puesto que el mensaje está cifrado con la llave pública que Patrick sustituyó, éste puede descifrarlo con su llave privada, luego reencriptarlo con la llave pública de Florencia y enviárselo.

Incluso en el caso de que las llaves públicas de Florencia y Francisco estén almacenadas en una base de datos, este ataque puede funcionar ya que Patrick puede interceptar la petición de información que Florencia le hace a la base de datos y substituir la llave pública de Francisco por la suya. El intruso en el medio, puede hacer la misma jugada con Francisco o más aún, puede irrumpir subrepticiamente en la base de datos y sustituir su llave propia por la de Florencia y Francisco. Luego simplemente espera a que éstos se comuniquen, intercepta sus mensajes y los sustituye realizando así su tarea.

Este tipo de ataque funciona debido a que tanto Florencia como Francisco no tienen forma de verificar que están hablando uno con el otro, es decir no tienen manera de corroborar la identidad del otro. Si la intrusión de Patrick no produce ningún retraso perceptible en las comunicaciones ninguno de los dos afectados tendrá idea de que hay alguien sentado en medio de ambos leyendo las comunicaciones que ellos piensan secretas.

El protocolo interseguro o de "interlock" inventado por R. Rivest y A. Shamir fue pensado para prevenir un ataque del tipo **hombre en medio del camino** [23] y funciona de la siguiente manera:

1. Florencia y Francisco intercambian sus llaves públicas.
2. Florencia cifra su mensaje empleando la llave pública de Francisco y le envía a él la mitad del mensaje cifrado.
3. Francisco cifra su mensaje empleando la llave pública de Florencia y le envía a ella la mitad del mensaje cifrado.
4. Florencia le envía la otra mitad del mensaje cifrado a Francisco.
5. Francisco junta las dos mitades del mensaje de Florencia y las descifra con su llave privada. Luego le envía a Florencia la otra mitad de su mensaje cifrado.
6. Florencia junta las mitades del mensaje de Francisco y las descifra con su llave privada.

El protocolo está diseñado para dificultar la tarea a un atacante que intente intervenir una comunicación mientras ésta sucede. El punto importante aquí es que una mitad del mensaje es inútil sin la otra, ya que no puede ser descifrado. ¿cómo es que ésto le causa problemas al intruso Patrick?

Para responder a esta pregunta, revisemos primero los pasos que éste debe realizar para atacar el protocolo. Aunque Patrick puede sustituir las llaves públicas de Florencia y Francisco por la suya en el paso 1, deberá además interceptar la mitad del mensaje de Florencia en el paso 2. Sin embargo, no podrá descifrar dicha mitad con su llave privada y reencriptarla de nuevo con la llave pública de Francisco, por lo que deberá crear un mensaje completamente nuevo y enviarle la mitad de éste a Francisco. Cuando Patrick intercepte la mitad del mensaje que Francisco le envía a Florencia en el paso 3, tendrá el mismo problema y deberá inventar un mensaje totalmente nuevo y enviarle la mitad a Florencia. Al momento de interceptar las segundas mitades de los mensajes en los pasos 4 y 5, será demasiado

tarde para cambiar los mensajes que inventó y envió, por lo que la conversación entre Florencia y Francisco será incoherente, por lo que ambos podrán deducir que sus comunicaciones están siendo atacadas. Ahora bien, con la información que Patrick obtuvo de escuchar las comunicaciones durante algún tiempo, podría, eventualmente, intentar simular ambos lados de la conversación entre las partes, las cuales podrían ahora no darse cuenta del engaño.

Intercambio de llaves con firmas digitales

Otra forma de evitar un ataque del tipo hombre en medio del camino es instrumentando las firmas digitales en el protocolo de intercambio de las llaves de sesión. En este esquema, el árbitro Ángel firmará las llaves públicas de Florencia y Francisco, dichas llaves firmadas incluirán un certificado de propiedad. De tal manera que cuando ambos reciban las llaves, cada uno de ellos podrá verificar la firma del árbitro y sabrá que la llave pública que recibe pertenece específicamente a la otra persona. Después se llevará a cabo el intercambio de llaves de sesión. De esta forma las dificultades para Patrick aumentan, ya que no puede personificar o suplantar a Florencia o Francisco porque no conoce sus llaves privadas. En este caso, Patrick no puede sustituir su llave pública por cualquiera de las de ellos, ya que la firma del árbitro que ésta lleva lo identifica como Patrick. Lo más que éste puede hacer es observar el tráfico de mensajes cifrados entre las partes.

Este protocolo emplea al árbitro y el riesgo de comprometer al KDC es menor que en el protocolo interseguro. Si de alguna manera Patrick logra irrumpir en el KDC y comprometer al árbitro, lo que obtendrá es la llave privada de éste, permitiéndole firmar llaves nuevas únicamente, pero impidiéndole descifrar las llaves de sesión o leer el tráfico de mensajes. Para leer el tráfico de mensajes Patrick tendrá que personificar a un usuario de la red y engañar a los demás usuarios para que cifren sus mensajes con esta llave pública fraudulenta. Si lanza este tipo de ataque, empleando la llave privada del árbitro, el intruso puede crear llaves firmadas fraudulentas para engañar a los usuarios legítimos. Luego, puede intercambiar en la base de datos las llaves firmadas legítimas por las que creó, o interceptar las peticiones que los usuarios hacen a la base de datos respondiéndoles con llaves fraudulentas lo que le permitiría lanzar un ataque de *hombre en el medio del camino* y leer así las comunicaciones de los usuarios.

Como mostramos, un ataque del tipo *hombre en el medio del camino* es posible aunque presenta varias dificultades. El que lanza el ataque debe ser capaz de interceptar y modificar mensajes. En algunas redes esto es mucho

más difícil que solamente escuchar el tráfico de mensajes. En algunos canales de transmisión, como en las redes de comunicación que emplean radios, es casi imposible reemplazar un mensaje por otro a menos que la red completa sea interferida. En el caso de las redes de computadora, esto resulta más simple ya que el ataque se realiza sobre los ruteadores o suplantado las IPs.

5.3. Un protocolo híbrido

Para poder comunicarse entre sí, Florencia y Francisco no tienen necesariamente que completar el protocolo de intercambio de llaves antes de intercambiar mensajes. En el siguiente protocolo, Florencia le envía a Francisco un mensaje M, sin haber realizado previamente el protocolo de intercambio de llaves:

1. Florencia genera aleatoriamente una llave de sesión K y cifra con ésta el mensaje M: $E_F(M)$.

2. Florencia obtiene la llave pública de Francisco de una base de datos.

3. Florencia cifra la llave de sesión K empleando la llave pública de Francisco: $E_{Fr}(K)$.

4. Florencia le envía a Francisco tanto el mensaje cifrado como la llave de sesión encriptada: $E_F(M)$, $E_{Fr}(K)$. Para aumentar la seguridad contra un ataque de *hombre en el medio del camino*, Florencia firma la transmisión.

5. Francisco descifra la llave de sesión K empleando su llave privada y descifra el mensaje M empleando esta llave.

Este sistema híbrido es utilizado en la mayoría de los sistemas de comunicación que emplean la criptografia de llave pública, ya que puede ser combinado con otros protocolos de seguridad como las firmas digitales y las estampas temporales. Para finalizar esta discusión sobre intercambio de llaves, dejamos abiertas dos preguntas: ¿qué pasa en el caso de que Florencia tenga que enviar el mensaje cifrado a varias personas? y ¿qué protocolo debe establecer para realizar esta tarea de manera segura?

5.4. Autenticación

Cuando Florencia accede a una red de comunicaciones a través de una computadora o un teléfono bancario ¿cómo ese sistema reconoce su identidad y verifica que efectivamente es ella y no otra persona la que solicita el acceso suplantándola? La manera tradicional de hacerlo es empleando contraseñas: Florencia ingresa la contraseña y si es correcta, el sistema le permite seguir adelante. En este esquema, tanto Florencia como el sistema conocen esta pieza secreta de conocimiento. Cada vez que alguien quiera ingresar al sistema, éste le solicitará la contraseña. Sin embargo, existen diferentes maneras de realizar la autenticación de la identidad de un usuario; en esta sección estudiaremos algunas de ellas.

Autenticación empleando funciones de una sola vía

Algunos protocolos suponen que es necesario que el host del sistema conozca todas las contraseñas, sin embargo existen otros protocolos en los que éstas no son almacenadas. En este tipo de protocolos, el sistema[1] no necesita conocer las contraseñas, sino que es suficiente que pueda discernir entre las contraseñas válidas y las que no lo son. Esto se puede lograr empleando funciones de una sola vía. De tal manera que, en lugar de almacenar las contraseñas, es suficiente almacenar las funciones de una vía generadas por dichas contraseñas. A continuación se presenta un protocolo que ilustra esta idea.

1. Florencia le envía al sistema su contraseña.

2. El sistema obtiene el valor hash a partir de una función de una vía sobre la contraseña.

3. El sistema compara este valor hash con los previamente almacenados.

De esta manera, si un intruso roba la base de datos, lo único que tiene son los valores hash de las contraseñas. Dichos valores resultan inútiles al intruso ya que es muy difícil recuperar las contraseñas a partir de ellos.

[1]Sistema, host del sistema o simplemente host: es el encargado de mediar los múltiples accesos a la base de datos y el responsable de proveer los servicios del sistema.

Ataques de diccionario

Un archivo de contraseñas cifrado con una función de una sola vía sigue siendo relativamente vulnerable. Pensemos que Patrick dispone de los recursos necesarios para recabar una lista que contiene las contraseñas más empleadas. Luego les aplica a todas ellas una determinada función de una sola vía y almacena los resultados. Después de esto, Patrick roba el archivo de los valores hash del sistema que intenta penetrar y lo compara con valores hash de las posibles contraseñas que posee para encontrar las coincidencias. Un ataque de este tipo es conocido como **ataque de diccionario** y resulta bastante exitoso.

Una técnica para dificultar el ataque de diccionario consiste en concatenar a las contraseñas una cadena de bytes, llamada valor de *salt* (sal), antes de aplicarles la función de una vía. Después se almacenan en la base de datos del sistema los valores de dicha cadena y los resultados obtenidos al aplicar la función de una vía. Si el número de posibles valores de *salt* es suficientemente grande, un ataque de diccionario se dificulta ya que sería necesario generar el valor de hash de cada posible valor de la cadena *salt* concatenada a la contraseña. El punto aquí es asegurarse de que un intruso como Patrick tenga que realizar un número muy alto de ensayos para el cifrado de cada contraseña contenida en su diccionario cada vez que intente romper la seguridad de la contraseña de otra persona.

La técnica mencionada resulta útil para protegerse contra ataques de diccionario realizados sobre archivos de contraseñas, pero es muy débil en un ataque dirigido sobre una sola contraseña. Podría también resultar de cierta utilidad para aquellos que emplean la misma contraseña en varios sistemas. Sin embargo, es importante recordar que las estampas temporales son un elemento que se puede emplear para mejorar la seguridad de ciertos sistemas.

Daniel Klein desarrolló un programa que adivina alrededor del 40 % de las contraseñas almacenadas en un sistema en tan solo una semana [21]. Por otro lado D. Feldmeier y P. Karn recopilaron una lista de alrededor de 700,000 contraseñas comúnmente empleadas, concatenadas con cada uno de los 4096 valores *salt* posibles [22]. La estimación es que cerca del 30 % de las contraseñas de cualquier sistema estándar pueden ser rotas empleando esta lista y conociendo las funciones hash más frecuentemente empleadas por estos sistemas.

Autenticación empleando criptografía de llave pública

Cuando Florencia le envía su contraseña en claro al sistema, cualquiera que tenga acceso a la vía de comunicación puede interceptar la contraseña. Es muy posible que ella acceda al sistema a través de una transmisión con un camino complejo y que quizá pase por varias instituciones o países extranjeros. Un intruso pasivo como David podría estar en cualquiera de esos puntos escuchando la secuencia de acceso de Florencia. Incluso, si David lograra acceso a la memoria del procesador del host, podría acceder a la contraseña antes de que se le aplique la función hash.

La criptografía de llave pública puede resolver este tipo de problemas en la seguridad de las contraseñas. Supongamos que el sistema conserva un archivo con las llaves públicas de todos los usuarios y éstos a su vez guardan sus propias llaves privadas. Un primer intento de protocolo sería el siguiente:

1. El sistema le envía a Florencia una cadena aleatoria.
2. Florencia cifra la cadena con su llave privada y la envía de regreso al sistema junto con su nombre.
3. El sistema busca la llave pública de Florencia en su base de datos y descifra el mensaje empleando dicha llave.
4. Si la cadena descifrada concuerda con la que el sistema originalmente le envió a Florencia, entonces éste le permite el acceso.

Ya que nadie más que Florencia tiene acceso a su llave privada, no es posible suplantarla. Además de que Florencia nunca le envía al sistema su llave privada. El intruso David, que monitorea la interacción, no tiene manera de obtener información que le permita deducir la llave privada de Florencia y suplantarla.

Recordemos que es mala idea enviar cadenas arbitrarias cifradas a un tercero poco confiable. En ciertas circunstancias resulta un grave error de seguridad cifrar cualquier tipo de cadenas que nos han sido enviadas, ya que pueden abrir la puerta a diferentes ataques. Un protocolo seguro de prueba de identidad toma una forma más complicada que la del protocolo previo:

1. Florencia genera una cadena a partir de una serie de cálculos sobre números aleatorios y su llave privada. Luego, envía esta cadena al sistema.
2. El sistema le envía a Florencia otro número aleatorio.

3. Florencia genera otra cadena a partir de una serie de cálculos sobre los números aleatorios que ella generó así como sobre los que recibió del sistema y sobre su llave privada. Le envía los resultados al sistema.

4. El sistema genera otra cadena a partir de algunos de los números que recibió de Florencia y sobre la llave pública de ella para verificar que Florencia conoce su propia llave privada, confirmando así su identidad.

En el caso de que Florencia no confíe en el sistema, tanto como éste desconfíe de ella, puede solicitarle que pruebe su identidad de la misma manera.

Apéndice A

Comenzando con *Python*

Para escribir nuestros programas en *Python* usaremos el programa de edición *IDLE* (Entorno de desarrollo y aprendizaje integrado, por las siglas de *Integrated Development and Learning Environment*), el cual se incluye en el paquete de instalación que puede descargarse de la página oficial (`www.python.org`). Al ejecutar el programa de edición, aparecerá la ventana llamada *Shell*:

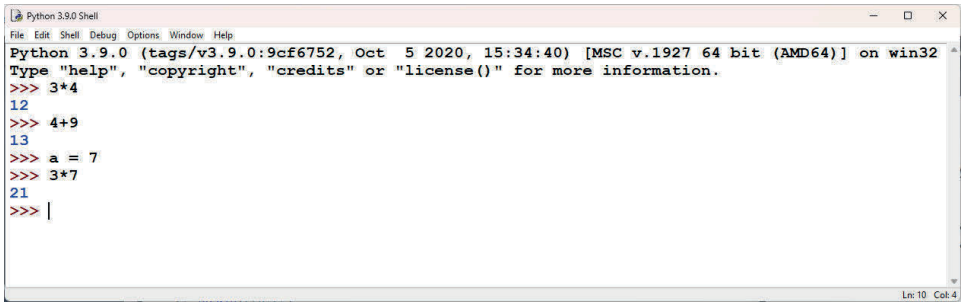


Figura A.1: Ventana de *Shell*.

En esta ventana podremos visualizar la salida de los programas. Además es posible realizar cálculos sencillos, pero para comenzar a escribir un programa, debes pulsar *File* y *New File*.

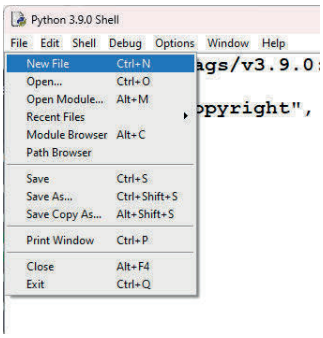


Figura A.2: Seleccionar *File* y *New File* para abrir la ventana de edición.

Con lo que tendremos una ventana de edición:

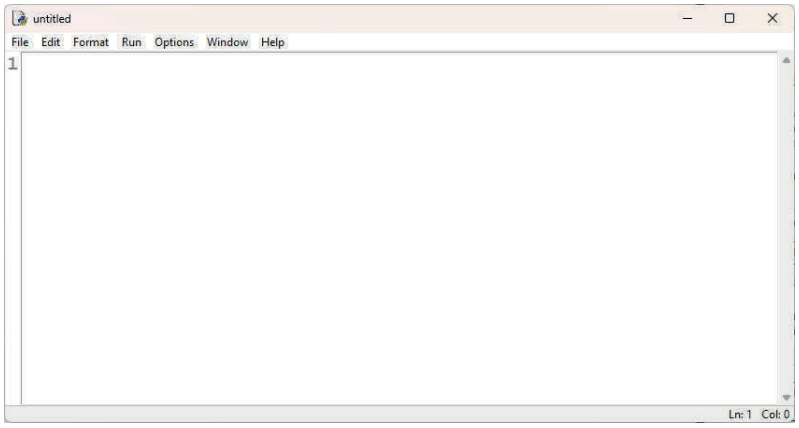


Figura A.3: Ventana de edición.

La estructura de un programa en *Python* es muy sencilla. Comenzaremos por mostrar el uso de la función `print`, la cual muestra un mensaje en la ventana *shell*:

```
print('Mi primer programa en Python')
```

De forma similar a cuando programamos en *C*, podemos mostrar el valor de una variable de la siguiente manera:

```
a = 5
print('El valor de a es%d' % a)
```

Con lo que el programa mostrará el mensaje:

```
El valor de a es 5
```

Notamos que en *Python*, la declaración de la variable es intrínseca a la asignación, en el programa anterior la variable `a` es de tipo entero. Sin embargo, al hacer la asignación de la siguiente manera:

```
a = 5.0
print('El valor de a es%f' % a)
```

la variable `a` es de tipo flotante, y habrá que manejarla como tal al mostrarla con la función `print`

Las operaciones entre variables son muy intuitivas. Veamos el siguiente programa:

```
b = 5.0
h = 3.0
a = b*h/2
print('El área de un triangulo con base%f y altura%f es%f' %
(b,h,a))
```

Notamos que al mostrar varias variables con `print`, es necesario agruparlas en una tupla, en el orden en que se muestran en el mensaje. El programa mostrará:

```
El área de un triángulo con base 5.0 y altura 3.0 es 7.5
```

Ahora bien, si deseamos que se le soliciten al usuario los datos del triángulo, empleamos la función `input` que recibe como argumento el mensaje con el que se solicitará el dato al usuario y devolverá el valor que se introduzca en el teclado. Por ejemplo:

```
b = input('Dame el valor de la base ')
```

de este modo, la variable `b` tomará el valor que el usuario introduzca. Sin embargo este valor será siempre una cadena de texto, de modo que para poder hacer operaciones, debemos convertir esta cadena de texto en el tipo de valor que requiramos:

```
b = float(input('Dame el valor de la base '))
```

Así, el programa quedaría de la siguiente forma:

```
b = float(input('Dame el valor de la base '))
h = float(input('Dame el valor de la altura '))
a = b*h/2
print('El área de un triangulo con base%f y altura%f es%f'%
(b,h,a))
```

Otro ejemplo podría ser el cálculo del volumen de una esfera:

```
r = float(input('Dame el radio de la esfera '))
v = (4/3)*3.1416*r**3
print('El volumen de una esfera de radio%f es%f'% (r,v))
```

Notamos que la forma de elevar un número a alguna potencia (x^y), en *Python* se denota como `x**y`.

Las listas son una estructura fundamental en la programación con *Python*. Podemos crear una lista de valores asignando directamente una colección de éstos, separados por comas y delimitados por corchetes. Por ejemplo:

```
A = [3,0,'k',0.5,1,'python']
```

Podemos identificar al primer elemento como `A[0]`, al segundo como `A[1]`, etc. e incluso podemos referirnos al último término como A[-1], al penúltimo como A[-2], etc. de modo que:

```
print(A[0]])
```

mostrará el número 3, y

```
print(A[-1]])
```

mostrará `python`. También podemos aumentar elementos a la lista con el método `append`, como sigue:

```
A = [3,0,'k',0.5,1,'python']
A.append(7)
```

Con lo cual la lista quedará como `A = [3,0,'k',0.5,1,'python',7]`

La función `range`, devuelve una lista con el número de elementos que recibe como argumento. Por ejemplo:

```
A = range(5)
```

asigna a `A` la lista `[0,1,2,3,4]`

```
A = range(2,7)
```

asigna a `A` la lista `[2,3,4,5,6]`

```
A = range(3,10,2)
```

asigna a `A` la lista `[3,5,7,9]`.

Las listas en *Python* cobran aún más relevancia al ser indispensables para la estructura del ciclo `for`. La instrucción `for` repite la ejecución de las líneas que le siguen y que están alineadas en una sangría (indentación) mayor, indicando que estarán contenidas en el cuerpo del ciclo. En el programa se muestra esta sangría en color azul:

```
for k in [3,5,7,9]
    print('El valor de k es:')
    print(k)
```

En el programa anterior, las líneas que contienen la sangría azul, se repiten una vez por cada elemento de la lista [3,5,7,9], y la variable `k` toma el valor de cada elemento, así el programa mostrará:

```
El valor de k es:
3
El valor de k es:
5
El valor de k es:
7
El valor de k es:
9
```

En ocasiones, es necesario crear listas más especiales, como por ejemplo, una lista que contenga 100 *ceros*. Sería muy tedioso crear la lista dando directamente los valores, así que podemos recurrir también al ciclo `for` con la siguiente estructura:

```
B = [0 for i in range(100)]
```

De este modo, tendremos que el *cero* se "replica" dentro de la lista, teniendo una lista con 100 *ceros*. En general, esta estructura, replica una expresión que puede (o no) depender de la variable involucrada en el `for`:

```
B = [expresión(k) for k in Lista]
```

Otro ciclo es el `while` en este caso se repite el conjunto de instrucciones *mientras* se cumpla una condición, nótese que al comparar dos valores o variables se usa doble signo igual (`==`), mientras que al asignar un valor a una variable, usamos uno solo (`=`).

```
respuesta = 'si';
while (respuesta=='si'):
    input("Quieres que continue el ciclo (si,no)")
```

```
Quieres que continue el ciclo (si,no) si
Quieres que continue el ciclo (si,no) si
Quieres que continue el ciclo (si,no) si
Quieres que continue el ciclo (si,no) no
```

También es posible condicionar la ejecución de ciertas líneas mediante la instrucción `if`, como en el ejemplo siguiente:

```
a = 5;
b = 8;
if(b>a):
    print('La condición se cumple')
printf('FIN')
```

```
La condición se cumple
FIN
```

También es posible incluir operadores lógicos *or* y *and*:

```
a = 5
b = 8
c = 2
if(b>a and c<a):
    print('La condición se cumple')
print('FIN')
```

```
La condición se cumple
FIN
```

```
a = 5
b = 8
c = 2
if(b==8 or a==0):
    print('La condición se cumple')
print('FIN')
```

```
La condición se cumple
FIN
```

Puedes agregar un bloque de instrucciones que se ejecutarán en el caso de que no se cumpla la condición del `if`, agregando un bloque `else`:

```
a = 15
b = 8
if(b>a):
    print('La condición se cumple')
else:
```

```
    print('La condición no se cumple')
print('FIN')
```

```
La condición no se cumple
FIN
```

Además de las listas, *Python* incorpora una estructura muy útil; los *diccionarios*. Como vimos anteriormente, cuando definimos una lista, cada uno de sus elementos queda identificado por un número entero, es decir, por su índice. En el caso de los diccionarios, podemos identificar algún elemento (valor) dentro de él con una etiqueta o llave, de la siguiente manera:

```
D1 = {'A': 23, 'E': 48, 'I': 84, 'O': 14, 'U': 21}
```

de tal modo que `D['A']` tomará el valor de 23, `D['E']` el de 48, etc.

Es interesante que tanto el valor como su llave asociada pueden ser cualquier tipo de variable, por ejemplo, ambos pueden ser cadenas de texto:

```
D2 = {'uno': 'one', 'dos': 'two', 'tres': 'three',
        'cuatro': 'four', 'cinco': 'five'}
```

así, `D['tres']` tomará el valor de `'three'`.

Cabe mencionar que un diccionario puede definirse directamente, como lo hemos hecho, es decir, agrupando cada definición *llave*:*valor* entre corchetes y separados por comas, o bien agregando pares a un diccionario ya creado, por ejemplo, podemos comenzar por un diccionario vacío:

```
D3 = {}
```

e ir agregando definiciones una a una:

```
D3['A'] = 23
D3['E'] = 48
    ⋮
  etc.
```

Otra estructura muy útil son los *conjuntos*; tomemos por ejemplo la lista siguiente:

```
canasta = ['manzana', 'naranja', 'manzana', 'pera', 'naranja',
                          'platano']
```

y con ella podemos obtener un conjunto:

```
fruta = set(canasta)
```

o bien podemos definirlo directamente agrupando elementos entre corchetes y separados por comas:

```
fruta = {'manzana', 'naranja', 'manzana', 'pera', 'naranja',
                          'platano'}
```

entonces, podemos visualizar los elementos del conjunto con `print(fruta)`, con lo que obtendremos:

```
{'pera', 'manzana', 'naranja', 'platano'}
```

es decir, los elementos sin repeticiones y sin un orden establecido. De este modo, podemos consultar si un elemento pertenece a un conjunto, por ejemplo, `print('naranja' in fruta)`, mostraría `True`, y `print('mora' in fruta)`, mostraría `False`.

También podemos operar conjuntos, por ejemplo, si definimos el siguiente conjunto:

```
roja = {'manzana','cereza','fresa'}
```

podemos calcular y mostrar la diferencia de ambos conjuntos:

```
print(fruta - roja)
```

que mostraría {'pera', 'platano', 'naranja'}. O la unión

```
print(fruta | roja)
```

que mostraría {'cereza', 'fresa', 'pera', 'platano', 'naranja', 'manzana'}, para la intersección:

```
print(fruta & roja)
```

que mostraría {'manzana'}, y la diferencia simétrica (la diferencia de la unión y la intersección):

```
print(fruta ∧ roja)
```

cuyo resultado es {'naranja', 'fresa', 'pera', 'platano', 'cereza'}.
También podemos comparar conjuntos:

```
A = fruta ∧ roja
B = (fruta | roja) - (fruta & roja)
print('Son iguales?', A==B)
```

que mostraría:

```
Son iguales? True
```

A.1. Programas del libro

El siguiente código QR contiene una liga a una carpeta en el sitio *GitHub*, donde pueden descargarse los programas presentados a lo largo de este libro:

Índice de programas

Bibliografía

[1] W. Friedman, *The Index of Coincidence and Its Applications in Cryptography*, Aegean Park press, 1987.

[2] W. Friedman, *Methods for the Solution of Running-Key Chiphers*, Riverbank Publications, Riverbank Labs., 1918.

[3] S. Singh, *Los códigos secretos*, Debate, 2000.

[4] E.A. Poe, *Narraciones Extraordinarias*, Porrúa, 2019.

[5] A. Kerckhoffs, *La cryptographie militaire*, Journal des sciences militaires, vol. IX, pp. 5–83, Jan. 1883, pp. 161–191, Feb. 1883. Disponible en Internet.

[6] B. Schneier, *Applied Cryptography: Protocols, Algorithms and Source Code in C*, John Wiley, 1996.

[7] B. Schneier, *Data Guardians*, MacWorld, **V. 10** (2) 1993.

[8] A. Menezes, P. Oorschot y S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1998

[9] E.G. Moore, *Cramming more components into integrated circuits*, Electronics, **V. 38** (8) 1965. Disponible en Internet.

[10] D. Khan, *The Codebreakers: The Story of Secret Writing*, New York, Macmillan Públishing Co., 1967.

[11] M. Gardner, *Codes, Ciphers and secret writing*, Dover publications, 1972

[12] H. Gaines, *Cryptanalysis. A study of ciphers and thier solution*, Dover Publications, 1956. Disponible en Internet.

[13] F. Pratt, *Secret and Urgent. The Story of Codes & Ciphers*, Blue Ribon Books, 1942.

[14] P. Wayner, *Mimic functions*, Cryptologia **V. 16** (3), 1992.

[15] W. Barker, *Cryptanalysis of the Hagelin Cryptograph*, Aegean Park press, 1977.

[16] W. Friedman, *Elements of Cryptanalysis*, Aegean Park press, 1976.

[17] W. Diffie y M Hellman, *New Directions in Cryptography*, IEEE Transactions on Information Theory, IT-22, n. 6, 1976.

[18] W. Diffie, *The First Ten Years of Public Key Cryptography*, Contemporary Cryptology: The Science of Integrity, G. Simmons, IEEE Press, 1992.

[19] N. Stephenson, *Criptonomicón*, De bolsillo, 2005.

[20] Khonfelder, *Towar A Practical Public Key Criptosystem*, Tesis, MIT Departament of Electrical Engieneering, 1978.

[21] D. Klein, *Foling the Cracker: A Survey of, and Implications to, Password Security*, Proceedings of the USENIX UNIX Security Workshop, 1990.

[22] D. Feldmier y P. Karn, *Unix Password Security -Ten Years Later*, Advances in Cryptology Proceedings, Springer Verlag, 1990.

[23] R. Rivest y A. Shamir, *How to Expose an Eavesdroper*, Communications of the ACM, **V. 27** (4) 1984.

[24] G. Simmons, *A Weak Privacy Protocol Using the RSA Cryptosystem*, Cryptologia, **V. 7** (2) 1983.

[25] G. Simmons, *How to Insure tha Data Acquired to Verify Treaty Compliance Are Trustworthy*, Contemporary Cryptology: The Science of Integrity, G. Simmons, IEEE Press, 1992.

[26] C. Kline, *Encryption and Secure Computer Networks*, ACM Computing Surveys, **V. 11** (4) 1979.

[27] M. Wilkes, *Time-Sharing Computer Systems*, New York: American Elsevier, 1968.

[28] J. Feigenbaum, M. Lieverman y R. Wright, *Cryptographic Protection of Data Bases and Software*, Distributed Computing and Cryptography AMS, 1991.

[29] N. Ferguson y B. Schneier, *Cryptography Engineering: Desings, Principles and Practical Applications*, John Wiley, 2010.

[30] N. Ferguson y B. Schneier, *Practical Cryptography*, John Wiley, 2003.

[31] M. Garey, D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co. 1979.

[32] C. Deavours y L. Kruh, *Machine Cryptography and Modern Cryptanalysis*, Norwood MA: Artech House, 1985.

[33] D. Dolev, A. Yao, *On the Security of Public Key Protocols*, Proceedings of the 22nd Annual Symposium on the Foundations of Computer Science, 1983.

[34] R. Merkle, *Secure Comunication Over Insecure Channels*, Comunications of the ACM, **V. 21** (4) 1978.

[35] S. Akl, *Digital Signatures: A Tutorial Survey*, Computer, **V. 16** (2) 1983.

[36] S. Haber y W. Stornetta, *How to Time-Stamp a Digital Document*, Advances in Cryptology Proceedings, Springer Verlag, 1991.

[37] R. del Castillo, *Introducción al cómputo científico con Python*, Las Prensas de Ciencias, 2021.

[38] A. Sweigart, *Hacking Secret Ciphers with Python*, (under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States License), 2013.

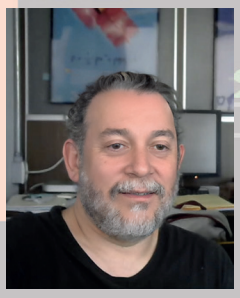
[39] Y. Desmedt y Y. Frankel, *Theshold Cryptosystem*, Advances in Cryptology Proceedings, Springer Verlag, 1990.

[40] `www.schneier.com`

[41] `https://www.schneier.com/wp-content/uploads/2015/01/schneier-talk.pdf`

Introducción a la criptografía
Explicaciones, algoritmos y códigos en Python
se terminó de imprimir en diciembre de 2025,
en el taller de impresión de la
Universidad Autónoma de la Ciudad de México,
San Lorenzo, 290, Col. Del Valle,
Alcaldía Benito Juárez, C. P. 03100,
Ciudad de México con un tiraje de 500 ejemplares.
Corrección de estilo y cuidado de la edición: Nancy Sanciprián
Diseño editorial: Sergio Cortés Becerril

La criptografía tiene una larga y fascinante historia. La utilización de técnicas criptográficas abarca más de 4,000 años de la historia humana y ha sido empleada tradicionalmente por militares, bancos, comerciantes, servicios diplomáticos, y en general por los gobiernos desde tiempos antiguos para asegurar sus comunicaciones y proteger sus secretos. La criptografía está conformada por un conjunto muy amplio de técnicas que tratan sobre la protección de la información y emplea diversos conceptos y técnicas matemáticas relacionadas con la confidencialidad, la integridad y la autenticidad de un mensaje.



Raúl A. Espejel Morales cursó sus estudios de licenciatura en física en la Facultad de Ciencias, obtuvo el grado de maestro en ciencias en el Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas y obtuvo el doctorado en el Instituto de Investigaciones en Materiales, todo esto en la Universidad Nacional Autónoma de México. Imparte clases de la carrera de física relacionadas con Computación Científica en la Facultad de Ciencias de la UNAM desde el año 1996 y es profesor de tiempo completo desde el año 2005. Sus intereses están relacionados con las Ciencias de la Computación y la instrumentación, así como la implementación de las nuevas tecnologías en la docencia.

Iván O. Sosa Pérez cursó sus estudios de licenciatura en física en la Facultad de Ciencias, obtuvo el grado de maestro en ciencias en el Instituto de Física de la Universidad Nacional Autónoma de México. Es profesor-investigador de tiempo completo desde 2003 en el Colegio de Ciencia y Tecnología de la Universidad Autónoma de la Ciudad de México (UACM). Sus intereses están relacionados con la Física Computacional, la Óptica y Física de superficies, así como el desarrollo de las nuevas tecnologías en la enseñanza.

UACM
Universidad Autónoma de la Ciudad de México
NADA HUMANO ME ES AJENO

Biblioteca del Estudiante

9 786072 615786