Fundamentos de Seguridad y Protección de Sistemas Informáticos

curso 2019/2020

francisco miguel garcía olmedo

7 de noviembre de 2020

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Breve Introducción Histórica

El término "criptografía" proviene de "crypto" y de "graphein". El significado previsible a partir de esta etimología sería "escritura oculta" o, más profundamente, "escritura en clave en la que se alteran los signos gráficos correspondientes a los fonemas de una lengua".

La criptografía o escritura cifrada fue ya empleada en la antigüedad y continúa siéndolo en nuestra época, especialmente por las cancillerías y los Estados mayores del Ejército, los cuales poseen sus métodos particulares para el despacho de los documentos secretos. La escritura secreta puede serlo por transposición y por sustitución. En el primer caso las letras se disponen de manera diferente de la normal y hay que conocer de antemano el orden adoptado para poder descifrar la escritura. Pertenece a este sistema la escritura secreta de los lacedemonios, los cuales escribían las letras sobre una correa en tal orden, que al ser aquella enrollada sobre un bastón de determinado grueso las letras aparecían unidas formando palabras y frases.

El sistema de sustitución no altera el orden natural de las letras, pero los signos usados en la escritura normal son sustituidos por otros. Este sistema se empleaba ya en tiempo de los romanos y ha continuado usándose hasta hoy. Los métodos de cifra de este sistema son variados y pueden estar basados en signos convencionales, letras o números.

Los espartanos (400 a.J.) también utilizaban sus propios métodos de cifrado. El "rabo del cielo" era un palito sobre el que se enrollaba una tira en espiral conteniendo el mensaje que al quedar así dispuesto tomaba sentido.

Aeneas Tacticus (360 a.J.), incluye en su manual de guerra un capítulo titulado "Sobre

los mensajes secretos", quedando de manifiesto en éste, como en otros casos, que la criptografía es hija de la guerra. Aeneas describe varios métodos de cifrado.

Julio Cesar usaba el cifrado

$$P \longrightarrow P + 3 \pmod{26}$$

para comunicarse con sus generales, motivo por el cual a la familia de cifrados

$$P \longrightarrow P + k \pmod{26}$$

para $0 \leq k < 26$, se le llama *Cifrado del Cesar* y hoy día, o hasta hace muy poco, ha sido base de sistemas más complejos.

Durante la Edad Media la escritura cifrada se utilizó poco en España. En los siglos X y XII aparece en subscripciones y notas de algunos códices en letra visigótica, y consiste en el uso de ciertos signos derivados de la cursiva latina, en invertir la dirección de la escritura y en el uso de signos de otros alfabetos y de números. Estos sistemas a veces se utilizaban tan sólo para la sustitución de algunas letras o de algunas palabras. A fines de la Edad Media, desde el siglo XV, los códigos cifrados se perfeccionaron considerablemente y ya no se limitaron a sustituir ciertas letras como las vocales, para hacer más difícil la lectura, sino que abarcaban todo el texto. En las Repúblicas italianas el sistema se perfeccionó al sustituir una misma letra por varios signos y al emplear signos sin valor. Esto complicaba el desciframiento, pues disimulaba la frecuencia del uso de ciertas letras o de ciertas sílabas o palabras, lo cual constituye uno de los mejores recursos para llegar a descubrir la clave.

En la clave conservada en Siena, que había usado San Bernardino, cuyos viajes de predicación eran aprovechados con fines diplomáticos, la combinación de signos era muy variada:

- Un signo para cada consonante.
- Tres signos diferentes para cada vocal.
- Seis signos sin valor.
- Diez figuras (estrellas, rombos, cuernos de la luna, etc.) para los nombres propios.

Este sistema de pluralidad de signos para sustituir una misma letra obligaba al descifrador a una labor de tanteo mucho más laboriosa para descubrir la clave.

Uno de los primeros tratadistas de este sistema en Europa fue el alemán *Juan de Trittenheim* o *Trithemius* en su *Poligraphia* (1502). El método de *Trithemius* fue

muy perfeccionado por el italiano *Porta* en *De furtivis litterarum notis* (1563) y llevado a la perfección por el francés *François-Blaise de Vigenère*, autor del *Traité des Chiffres ou Secrètes Manières d'Écrire* (1586).

En la Edad Media queda, en muchas ocasiones, asociada a disciplinas esotéricas. *Tritemio* con su *Steganographia, hoc est ars per occultam scripturam animi sui voluntatem absentibus aperiendi certa* (Frankfort, 1606) —El arte de abrir, através de la escritura secreta, nuestra alma a las personas lejanas— recopila muchas de las técnicas existentes hasta entonces.

Tritemio, abad benedictino en Spannheim que vivió entre los siglos XV y XVI, miembro de una sodalitas céltica, que cultivaba la filosofía, astrología y la matemática pitagórica es a su vez conocedor del hebreo, caldeo y tártaro, y estaba en contacto con teólogos, cabalistas y alquimistas.
Dice que el destinatario deberá evocar ángeles como *Pamersiol*, *Padiel*, *Dorothiel*, etc., pero en realidad los ejemplos que contiene son militares y el libro está dedicado a *Felipe* duque de Baviera. Presenta cuarenta criptosistemas mayores y diez menores. El utilizado por *Umberto Eco* en su obra *El péndulo de Foucault* es el más simple:

$$P \longrightarrow P + 21 \quad (\text{mód } 22)$$

El código cifrado más antiguo en España es el de *Puebla*, para su correspondencia con los *Reyes Católicos*. Consiste en un breve diccionario de 2.400 palabras, en el que cada letra, sílaba o palabra ha sido sustituida por un numeral romano.

Al subir al trono, *Felipe II* ordenó la confección de nuevas claves, porque la cifra que usaba *Carlos V* para comunicarse con sus ministros de Italia y con otras personas "había envejecida y estaba ya muy divulgada". *Felipe II* modificó esta cifra en diferentes ocasiones. En el *Archivo de Simancas* existe una colección de cifras de los siglos XVI y XVII, conservada en uno de los legajos de la Secretaría de Estado.

Casanova juega con la criptografía para conseguir "rendidas pasiones".

Edgar Allan Poe, que gustaba de darse fama a sí mismo de criptoanalista, escribió una vez en una columna de un periódico de Filadelfia (Alexander's Weekly Messenger) que él podía solucionar cualquier criptograma (sustitución monoalfabética). *G. W. Kulp* remitió un texto cifrado de 43 palabras. En la misma columna, decía días más tarde que el citado texto era una "jerga incomprensible de caracteres aleatorios sin ningún sentido". En 1975, el criptograma de *Kulp* ha sido roto por *B.I. Winkel* y *M. Lyster*, demostrando que contenía 15 errores, razón por la que aparecía evidente su dificultad. El mismo *Poe*, da una muestra de criptoanálisis de frecuencias en su obra *El Escarabajo de Oro*.

En la Primera Guerra Mundial, *E.B. Fleissner* y *A. Figl* publican manuales importantes sobre el tema. Una segunda entrega de *A. Figl*, por entonces jefe del grupo de cifra del Ministerio Federal para Asuntos de las Fuerzas Armadas de Alemania es censurada y prohibida por el propio Ministerio, que contemplaba como los sistemas usados por ellos eran descritos cuidadosamente en el libro. Esto hace avanzar la criptografía hacia nuevas metas. Se pretende llegar a sistemas criptográficos tales que la fortaleza sea independiente del conocimiento público del algoritmo que se utiliza. No obstante, este sigue siendo un mundo de secretos oficiales y materias reservadas. Hoy en día no es conocido el método usado por la Agencia Nacional de Seguridad americana en sus transmisiones. Recientemente La NSA ha censurado la publicación de algunos artículos en criptografía y se han producido fricciones entre investigadores y la NSA. En 1980, la American of Education ACE ha recomendado un sistema de review voluntario por parte de los investigadores, contestado por muchos de estos como un ataque a la libertad intelectual y a la comunidad científica misma.

Una de las más espectaculares azañas realizadas por los criptógrafos fue el diseño y puesta en práctica de la *Máquina Enigma*. El sistema es esencialmente una máquina de rotores. Fue diseñada por *Arthur Scherbius* y *Arvid Gerhard Damm* en Europa y adoptada por los ejércitos alemán y japonés durante la Segunda Guerra Mundial. La ruptura del sistema fue encargada a un equipo de criptógrafos polacos, quienes tuvieron éxito con una máquina simplificada. El equipo se desplazó por Europa a medida que cambiaban las condiciones bélicas, llegando a estar en Francia y España. Finalmente correspondió a un equipo británico, en el que se encontraba *Alan Turing* la ruptura definitiva del sistema criptográfico.
A partir de la segunda guerra mundial, con los ordenadores y su expansión, un nuevo concepto en criptografía, basado en la complejidad computacional, se impone.

Las últimas etapas interesantes en la historia del secreto quizás sea el compromiso de los sistemas soviéticos de cifra después del derribo del jumbo de la KLA, o el conocimiento de los criptosistemas cubanos después de la invasión de la isla de Granada por parte de los estadounidenses. Recientemente todavía, y con un tinte más circense, los servicios secretos españoles se han visto sorprendidos con el conocimiento por parte de los marroquíes de unas conversaciones entre España y el Frente Polisario.

Desde otro punto de vista, métodos específicos de autentificación criptográfica se han desarrollado para el seguimiento de los acuerdos sobre limitación de armas estratégicas USA-URSS, por los laboratorios de Livermoore y los Alamos, dependientes de la V.C. Berkeley en lo que se ha llamado con el programa TNB (Nuclear Test Ban).

Recientemente, la criptografía adquiere nuevos enfoques, lejos del sentido militar. Cuestiones como la autentificación, la identificación, el acceso a redes de ordenadores o los protocolos de comunicación informáticos son nuevos campos de aplicación de la cripto-

grafía.

1.2. Vocabulario Básico.

Un *alfabeto* A es un conjunto finito no vacío de símbolos. Si A es un alfabeto, entonces:

$$exp(A) = \bigcup_{k \in \omega} A^k$$

es el cojunto de las expresiones construidas con símbolos del alfabeto A.

Si $x \in exp(A)$ es porque existe $k \in \omega$ tal que $x \in A^k$; entonces $x = \langle x_j \rangle_{j \in k}$, para unos ciertos elementos $x_j \in A$, y k es la *longitud* de x notada por $len(x)$.

Un *criptosistema*, *criptograma*, *cifrado* o *cifra* es una terna $\langle A, E, D \rangle$, donde A es un alfabeto y E (resp. D) es una aplicación del conjunto $exp(A)$ en $exp(A)$, tales que $D \circ E = 1_{exp(A)}$.

Los elementos de $exp(A)$ son denominados *texto llano* o *texto cifrado*; lo primero, si vienen como argumentos de E y lo segundo, si vienen como argumentos de D. Así pues, el elemento $E(x)$ (resp. $D(x)$) es denominado texto cifrado (resp. texto llano).

Dado un criptosistema $\langle A, E, D \rangle$, es muy frecuente que tanto E como D vengan dadas como la codificación de algoritmos, algoritmos que reciben el nombre de *criptográficos*. El algoritmo que codifica a E (resp. D) es denominado *algoritmo de cifrado* (resp. *algoritmo de descifrado*). Si la seguridad de un algoritmo es basada en el desconocimiento de su naturaleza, lo llamaremos *algoritmo restringido*. Los algoritmos restringidos tienen hoy sólo un interés histórico.

En el desarrollo actual de la teoría, lo usual es encontrar que el criptosistema está determinado por un algoritmo que, a su vez, depende de un parámetro elegible entre una gama. Cada caso concreto de parámetro es lo que se denomina una *llave* o *clave*. La seguridad del algoritmo de cifrado radica en la dificultad para averiguar la clave k que le corresponde. El rango de los posibles valores de la clave se llama *espacio de claves* o *llaves*.

El valor de la clave afecta a las funciones de cifrado y descifrado, por tanto podemos escribir:

$$E_k(P) = C$$
$$D_k(C) = P$$

Y si la clave de cifrado y descifrado es la misma, entonces según lo dicho:

$$D_k(E_k(P)) = P$$

Existen algoritmos en los que las claves de cifrado y descifrado van por pares. Es decir, la clave de cifrado k_e, es diferente de la de descifrado k_d. En este caso:

$$\begin{aligned} E_{k_e}(P) &= C \\ D_{k_d}(C) &= P \\ D_{k_d}(E_{k_e}(P)) &= P \end{aligned}$$

Existen dos formas generales de algoritmos con clave:

- algoritmos simétricos
- algoritmos de clave (o llave) pública.

Los *algoritmos simétricos* son algoritmos en los que la clave de cifrado k_e puede calcularse a partir de la clave de descifrado k_d y viceversa. En muchos de estos sistemas ambas claves coinciden. Como característica añadida tienen la de que la función de cifrado y descifrado son idénticas. En realidad, descifrar es cifrar un texto ya cifrado. Estos algoritmos también se llaman algoritmos de clave secreta y algoritmos de clave única. Requieren que el emisor y el receptor compartan y guarden en secreto una clave antes de comenzar a intercambiar mensajes. La seguridad de un algoritmo simétrico reside en la clave; la divulgación de la clave produce que cualquiera pueda cifrar y descifrar mensajes en este criptosistema. El cifrado y descifrado con un algoritmo simétrico lo notaremos:

$$\begin{aligned} E_k(P) &= C \\ D_k(C) &= P \end{aligned}$$

Los algoritmos simétricos pueden ser divididos en dos categorías:

- Los que actúan sobre el texto llano bit a bit, y a estos se le llama *algoritmos de flujo* o *cifrados en flujo*.
- Los que actúan sobre el texto llano en grupos de bits. Los grupos de bits se llaman *bloques*, y los algoritmos se llaman *algoritmos de bloque* o *cifrados en bloque*.

Los *algoritmos de clave pública* se sustentan sobre un esquema en el que se es parte de un círculo de usuarios y cada uno de ellos posee doble clave: una secreta y otra pública. La secreta o privada jamás debe trascender mientras que la pública puede viajar en abierto, incluso ser incluida en un listín. El cifrado para el usuario será hecho vía la clave pública y el descifrado sólo será posible vía la clave privada. La seguridad descansa en la imposibilidad de calcular la clave privada con del conocimiento de la pública.

El propósito primario de la criptografía es mantener el texto llano (o la clave o ambos) secreto para los que escuchan a escondidas —también llamados *adversarios, atacantes, intrusos, oponentes*, o simplemente *enemigos*. *Criptoanálisis* es la ciencia de recuperación del texto llano de un mensaje sin la clave. El criptoanálisis con éxito puede recuperar el texto llano o la clave. También puede centrarse en encontrar las debilidades de un criptosistema, lo que eventualmente conduce a los resultados anteriores.

Un intento de criptoanálisis es un *ataque*. Un ataque con éxito recibe el nombre de *método*. La ejecución de un ataque presupone que el *criptoanalista* posee los detalles del algoritmo criptográfico. Aunque esto no ocurre en los casos reales de criptoanálisis, es una suposición convencional del criptoanálisis académico.

Hay seis tipos generales de ataques criptoanalíticos, listados en orden de potencia. Cada uno de ellos asume que el criptoanalista tiene un conocimiento completo del algoritmo utilizado:

- *Ataque con texto cifrado*. En este ataque, el criptoanalista no tiene más material de trabajo que el texto cifrado con el criptosistema que quiere intervenir. La cantidad de texto cifrado que posee también está sujeta a hipótesis, aunque podemos suponer que tenemos tanto texto como requiramos.
- *Ataque con texto llano conocido*. El criptoanalista no sólo tiene acceso al texto cifrado de varios mensajes sino que conoce el texto llano de dichos mensajes.
- *Ataque con texto llano elegido*. El criptoanalista no sólo tiene acceso al texto cifrado y al texto llano asociado para varios mensajes, sino que puede elegir el texto llano a cifrar.
- *Ataque con texto llano elegido iterativamente*.
- Ataque con texto cifrado elegido
- Ataque con texto cifrado

Los criptosistemas pueden ser clasificados por su seguridad en: seguros, incondicionalmente seguros, computacionalmente seguros (o fuerte).

Para llevar a cabo muchos de los cifrados clásicos es elegido un alfabeto A de n caracteres, donde $n \neq 0$, el cual es puesto en biyección con el número natural n mediante una función $f\colon a \longrightarrow n$. Nos referiremos a esta situación en lo sucesivo sin más comentario. Si f es la biyección que estemos usando y $s \in \exp(A)$, $f(s)$ es la representación de $\langle f(s_j)\rangle_j$. Como ejemplo considérese el alfabeto latino de 41 caracteres, según muestra la tabla de la Figura 1.1.

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

(a) Biyección f

	0	g	7	n	14	u	21	1	28	8	35
a	1	h	8	o	15	v	22	2	29	9	36
b	2	i	9	p	16	w	23	3	30	,	37
c	3	j	10	q	17	x	24	4	31	.	38
d	4	k	11	r	18	y	25	5	32	;	39
e	5	l	12	s	19	z	26	6	33	:	40
f	6	m	13	t	20	0	27	7	34		

(b) Biyección g

Figura 1.1: Enumeraciones del alfabeto.

Capítulo 2

Criptosistemás Clásicos

2.1. Criptosistema Afín

El criptosistema afín apareció referido por primera vez en 1967 por David Kahn, en el periódico The Times. Se basa en un cifrado: por sustitución, monoalfabético y simétrico.

Definición 2.1.1. Sea $n \in \omega^*$ y A un alfabeto de n símbolos. Para la terna $\langle a, b, n\rangle$, que será denominada *clave afín*, donde $a \in n$ es un natural no nulo tal que $(a, n) = 1$ y $b \in n$, sea la función: $e_{\langle a,b,n\rangle} : n \longrightarrow n$ definida como:

$$e_{\langle a,b,n\rangle}(x) = (ax + b) \text{ mód } n \tag{2.1}$$

La función de *cifrado afín* sobre un alfabeto A de n símbolos según la clave $\langle a, b, n\rangle$ es $E_{\langle a,b,n\rangle} : exp(A)^* \longrightarrow exp(A)^*$ definida mediante:

$$E_{\langle a,b,n\rangle}(\langle s_j\rangle_j) = \langle((f^{-1} \circ e_{\langle a,b,n\rangle} \circ f)(s_j)\rangle_j$$

En la clave afín $\langle a, b, n\rangle$: a recibe el nombre de *constante de decimación*, b el de *constante de desplazamiento* y n el *orden*.

Teorema 2.1.1. *Sea $n \in \omega^*$ y A un alfabeto de n símbolos. El espacio de claves para el cifrado afín sobre ese alfabeto tiene cardinal $n\phi(n) - 1$, donde $\phi(n)$ es la función totient de Euler.*

Demostración. Basta saber que $\phi(n)$ es el número de números naturales k tales que $1 \leq k \leq n$ y $(k, n) = 1$ (en particular $\phi(1) = 1$). Además, debemos tener en cuenta que $\langle 1, 1, n\rangle$ no esconde el mensaje, por lo que no es clave afín. □

Teorema 2.1.2. *Sea $\langle a, b, n\rangle$ una clave afín. La función de $e_{\langle a,b,n\rangle}$ es biyectiva y su inversa es $d_{\langle a,b,n\rangle} : n \longrightarrow n$ definida por:*

$$d_{\langle a,b,n\rangle} = e_{\langle a',b',n\rangle} \tag{2.2}$$

donde, $a' \in n$ *y cumple* $a'a \equiv 1 \pmod{n}$ *(es decir,* $a' = a^{-1} \bmod n$*)* *y* $b' = (-a'b) \bmod n$.

Demostración. En efecto, sean $a' = a^{-1} \bmod n$), $b' = (-a'b) \bmod n$ y $x \in n$:

$$\begin{aligned} d_{\langle a,b,n\rangle}(e_{\langle a,b,n\rangle}(x)) &= (a'e_{\langle a,b,n\rangle}(x) - a'b) \bmod n \\ &= a'(e_{\langle a,b,n\rangle}(x) - b) \bmod n \\ &= a'(((ax + b) \bmod n) - b) \bmod n \\ &= a'(ax + b - b) \bmod n \\ &= aa'x \bmod n \\ &= x \bmod n \\ &= x \end{aligned}$$

y recíprocamente:

$$\begin{aligned} e_{\langle a,b,n\rangle}(d_{\langle a,b,n\rangle}(x)) &= (ad_{\langle a,b,n\rangle}(x) + b) \bmod n \\ &= (a((a'x - a'b) \bmod n) + b) \bmod n \\ &= (a(a'(x - b) \bmod n) + b) \bmod n \\ &= (aa'(x - b) + b) \bmod n \\ &= (x - b + b) \bmod n \\ &= x \bmod n \\ &= x \end{aligned}$$

□

Corolario 2.1.3. *Sea* $\langle a, b, n\rangle$ *una clave afín. La función de descifrado para dicha clave afín,* $D_{\langle a,b,n\rangle} : exp(A)^* \longrightarrow exp(A)^*$*, es* $E_{\langle a',b',n\rangle}$ *donde* $a' \in n$*,* $a' = a^{-1} \bmod n$ *y* $b' = (-a'b) \bmod n$.

Demostración. Basta utilizar, según el Teorema 2.1.2, que $e_{\langle a',b',n\rangle} \circ e_{\langle a,b,n\rangle} = 1_n$. □

Ejemplo 2.1.1. Como ejemplos clásicos figuran los siguientes:

1. *Criptosistema de César*: $n = 26$, $a = 1$ y $b = 3$.
2. *Criptosistema Lineal*: para cualquier $n \in \omega^*$ y a cumpliendo las condiciones requeridas, cuando elegimos $b = 0$.

El criptoanálisis de la cifra afín puede ser hecha a fuerza bruta o como sigue. Supongamos tener un texto que sabemos cifrado por medio de una función afín, que para fijar ideas

puede ser $E_{\langle a,b,n\rangle}$. Criptoanalizarlo supone encontrar $D_{\langle a',b',n\rangle}$. Si suponemos conocido el cardinal del alfabeto n (será un número superior al de letras distintas que aparecen en el mensaje cifrado) y la lengua en la que se escribió el mensaje llano, entonces procedemos como sigue:

1. Obtener una tabla de frecuencias de los símbolos en la lengua del mensaje. Esto es, una ordenación de los símbolos según la frecuencia con la que se repiten en los textos.

2. Hacer una tabla de frecuencias para los símbolos del texto cifrado relativa al mismo (la información será tanto más fiel cuando más extenso sea el texto cifrado).

3. Formar la parejas de símbolos $\langle t_1, e_1\rangle$ y $\langle t_2, e_2\rangle$, siendo t_1 (resp. t_2) el símbolo con más frecuencia (resp. el segundo símbolo más frecuente) en la lengua y e_1 (resp. e_2) el símbolo con más frecuencia (resp. el segundo símbolo más frecuente) en el texto cifrado.

4. Formar el sistema:
$$\begin{cases} e_1 a' + b' \equiv & t_1 \pmod{n} \\ e_2 a' + b' \equiv & t_2 \pmod{n} \end{cases} \tag{2.3}$$

   a) Si $d = e_1 - e_2$ tiene inverso módulo n, d^{-1}, entonces $a' = d^{-1}(t_1 - t_2)$ mód n y $b' = t_1 - e_1 a'$ mód n.

   b) Si no es el caso, descartamos uno de los símbolos del texto cifrado sustituyéndolo por el tercero con mayor frecuencia. Procediendo como antes obtenemos dos posibles valores para a' y b'.

Ejemplo 2.1.2. Como en español moderno las letras ordenadas por frecuencia de aparición son:

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

si las letras más frecuentes en el texto cifrado son 2 y r en ese orden, supondremos que 2 es el cifrado de e y r el de a. Seguidamente plantearíamos el sistema:

$$\begin{aligned} 29a' + b' &\equiv 5 \pmod{41} \\ 18a' + b' &\equiv 1 \pmod{41} \end{aligned}$$

Si restamos ambas congruencias queda $11a' \equiv 4 \pmod{41}$, de donde

$$\begin{aligned} a' &\equiv 11^{-1} \cdot 4 \pmod{41} \\ &\equiv 15 \cdot 4 \pmod{41} \\ &\equiv 19 \pmod{41}. \end{aligned}$$

Como $a' \equiv 19 \pmod{41}$, podemos encontrar ahora el valor de b', que será

$$\begin{aligned} b' &\equiv 5 - 29 \cdot a' \pmod{41} \\ &\equiv 5 - 18 \pmod{41} \\ &\equiv 28 \pmod{41} \end{aligned}$$

Esto nos hace pensar que la clave de cifrado sería la formada por $13 \equiv 19^{-1} \pmod{41}$ y $5 \equiv -13 * 28 \pmod{41}$, esto es, $\langle 13, 5 \rangle$. Para leer el mensaje que nos llegó cifrado:

```
2.eh9p27ep2fnrfsregggereu5:2wew9e.9es:2w2
```

cifraríamos éste, a su vez, mediante la clave encontrada $\langle 19, 28, 41 \rangle$, obteniendo el texto llano del que provenía:

```
el poder desgasta ... a quien no lo tiene
```

El criptoanálisis más efectivo es la fuerza bruta "perezosa" completada con un análisis de frecuencias y las pruebas: *chi-cuadrado* y *diferencias cuadradas*

La descripción de la prueba *chi-cuadrado* es como sigue. Considere un alfabeto A no vacío que conste de n elementos, y que C es un texto cifrado codificado usando símbolos de A. El texto sin formato P correspondiente a C también se codifica usando símbolos de A. Sea M un candidato a desciframiento de C, es decir, M es el resultado de intentar descifrar C usando una clave afín $\langle a, b, n \rangle$ que no es necesariamente la misma clave utilizada para cifrar en mensaje original y que acabó originando P. Suponga que para todo todo símbolo s de A, $F_A(s)$ es su probabilidad de frecuencia característica y que $F_M(s)$ es su (probabilidad de) frecuencia de mensaje con respecto a M. La distribución de probabilidad de frecuencia característica de un alfabeto es la distribución de probabilidad de frecuencia esperada para ese alfabeto.[1] La distribución de probabilidad de frecuencia de mensaje de M proporciona una distribución de la proporción de ocurrencias de caracteres sobre la longitud del mensaje. Se puede interpretar la probabilidad de frecuencia característica $F_A(s)$ como la probabilidad esperada, mientras que la probabilidad de frecuencia de mensaje $F_M(s)$ es la probabilidad observada. Si M es de longitud l, entonces la frecuencia observada del símbolo s de A es:

$$O_M(s) = F_M(s) \cdot l$$

mientras que la frecuencia esperada es:

$$E_A(s) = F_A(s) \cdot l$$

[1] Está basada en el estudio de una amplia colección de textos de los escritos en la lengua que se sustenta sobre ese alfabeto.

El grado chi-cuadrado de M o equivalentemente de la clave $\langle a, b, n\rangle$, en símbolos $R_{\chi^2}(M)$, viene dado por:

$$R_{\chi^2}(M) = \sum_{s\in A} \frac{(O_M(s) - E_A(s))^2}{E_A(s)}$$

El criptoanálisis por fuerza bruta produce para cada clave posible $\langle a, b, n\rangle$ un candidato a descifrado $M_{\langle a,b,n\rangle}$ para P. Consideremos el conjunto D_P definido como sigue:

$$D_P = \{\langle a, b, n, R_{\chi^2}(M_{\langle a,b,n\rangle})\rangle \colon a, b \in n, (a, n) = 1\}$$

Ordenaríamos los elementos de D_P de menor a mayor por la última componente y la clave secreta usada para cifrar P sería otenida de las tres primeras componentes de los primeros elementos resultantes de la ordenación, eventualmente el primero.

La descripción de la prueba de *diferencias cuadradas* es idéntica a la de la prueba chi-cuadrado, salvo que en lugar de usar R_{χ^2} usamos R_{RSS}, definida como sigue en ese mismo contexto:

$$R_{RSS}(M) = \sum_{s\in A} (O_M(s) - E_A(s))^2$$

El sistema Sagemath mantiene implementados algunos criptosistemas clásicos, entre ellos el afín. Esto permite el siguiente diálogo:

```
sage: A = AffineCryptosystem(AlphabeticStrings()); A
Affine cryptosystem on Free alphabetic string monoid on A-Z
sage: P = A.encoding("The affine cryptosystem generalizes the shift cipher.")
sage: P
THEAFFINECRYPTOSYSTEMGENERALIZESTHESHIFTCIPHER
sage: a, b = 9, 13
sage: C = A.enciphering(a, b, P); C
CYXNGGHAXFKVSCJTVTCXRPXAXKNIHEXTCYXTYHGCFHSYXK
sage: A.deciphering(a, b, C)
THEAFFINECRYPTOSYSTEMGENERALIZESTHESHIFTCIPHER
sage: A.deciphering(a, b, C) == P
True
sage: A.inverse_key(a,b)
(3, 13)
sage: A.random_key()
(11, 15)
sage: Plist = A.brute_force(C)
```

```
sage: Plist
{(1, 0): CYXNGGHAXFKVSCJTVTCXRPXAXKNIHEXTCYXTYHGCFHSYXK,
 (1, 1): BXWMFFGZWEJURBISUSBWQOWZWJMHGDWSBXWSXGFBEGRXWJ,
 (1, 2): AWVLEEFYVDITQAHRTRAVPNVYVILGFCVRAWVRWFEADFQWVI,
 (1, 3): ZVUKDDEXUCHSPZGQSQZUOMUXUHKFEBUQZVUQVEDZCEPVUH,
 (1, 4): YUTJCCDWTBGROYFPRPYTNLTWTGJEDATPYUTPUDCYBDOUTG,
 (1, 5): XTSIBBCVSAFQNXEOQOXSMKSVSFIDCZSOXTSOTCBXACNTSF,
...
sage: Rank_chi_2 = A.rank_by_chi_square(C,Plist)
sage: Rank_chi_2
((9, 13), THEAFFINECRYPTOSYSTEMGENERALIZESTHESHIFTCIPHER),
 ((7, 3), LDOUTTIHOEBKRLMGKGLOCYOHOBUXIPOGLDOGDITLEIRDOB),
 ((19, 22), OWLFGGRSLVYPIONTPTOLXBLSLYFCRKLTOWLTWRGOVRIWLY),
 ((11, 23), RTASPPIFAWNOJRUCOCRAQEAFANSBIDACRTACTIPRWIJTAN),
 ((9, 23), PDAWBBEJAYNULPKOUOPAICAJANWHEVAOPDAODEBPYELDAN),
 ((21, 10), MSNPGGLCNBADOMVTDTMNJZNCNAPQLWNTMSNTSLGMBLOSNA),
...
sage: Rank_square_diff = A.rank_by_squared_differences(C, Plist)
sage: Rank_square_diff
[((9, 13), THEAFFINECRYPTOSYSTEMGENERALIZESTHESHIFTCIPHER),
 ((11, 23), RTASPPIFAWNOJRUCOCRAQEAFANSBIDACRTACTIPRWIJTAN),
 ((5, 6), UOTRAAVETFGDSULNDNUTXHTETGRQVKTNUOTNOVAUFVSOTG),
 ((1, 19), JFEUNNOHEMRCZJQACAJEYWEHERUPOLEAJFEAFONJMOZFER),
 ((7, 21), BTEKJJYXEURAHBCWAWBESOEXERKNYFEWBTEWTYJBUYHTER),
 ((17, 8), SEHLGGDYHJUNWSXTNTSHZFHYHULADMHTSEHTEDGSJDWEHU),
...
```

Ejercicio 2.1.1. Haga lo siguiente:[2]

1. Implemente las funciones aritméticas necesarias para dar soporte al criptosistema afín.
2. Implemente el cifrado y descifrado del criptosistema afín, permitiendo la selección de un alfabeto arbitrario.
3. Implemente una versión optimizada del ataque a fuerza bruta.
4. Implemente el ataque por frecuencias según el método *chi-square*.
5. Implemente el ataque por frecuencias según el método *squared differences*.

[2] Sugerimos el uso de `Python` o `Haskell`.

2.2. Cifrado de Vigenère

El *cifrado de Vigenère* es en realidad el cifrafo de *Giovan Battista Belasso*, expuesto en su libro de 1533 titulado *La cifra del Sig*. Este cifrado es a su vez una modificación exitosa de la cifra de *Johannes Trithemius* de 1508 publicada en su *Poligraphia*.

Blaise de Vigenère publicó su descripción de un cifrado de autoclave parecido, pero más robusto, antes del reinado de Enrique III de Francia, en 1586. Más tarde, en el siglo XIX, la invención del cifrado dejó de atribuirse a Vigenère.

El cifrado Vigenère ganó una gran reputación por ser excepcionalmente robusto. Incluso el escritor y matemático Charles Lutwidge Dodgson (Lewis Carroll) dijo que el cifrado Vigenère era irrompible en el artículo "The Alphabet Cipher"para una revista de niños.

En 1917, la revista Scientific American afirmó que el cifrado Vigenère era imposible de romper. Esta reputación era inmerecida, considerando que el método Kasiski resolvió el cifrado en el siglo XIX, y que algunos criptoanalistas habilidosos pudieron romper ocasionalmente el cifrado en el siglo XVI.

El cifrado Vigenère es lo suficientemente simple si se usa con discos de cifrado. Los Estados Unidos de América, por ejemplo, utilizaron un disco de cifrado para implementar el cifrado Vigenère durante la Guerra Civil estadounidense. Los mensajes confederados fueron poco secretos, ya que los miembros de la Unión solían descifrar los mensajes.

Gilbert Vernam trató de arreglar el cifrado (creando el cifrado Vernam-Vigenère en 1918) pero, a pesar de sus esfuerzos, el cifrado sigue siendo vulnerable al criptoanálisis. (No confundir con el cifrado de Vernam).

Definición 2.2.1. Sea $n \in \omega^*$ y A un alfabeto de n símbolos. Una *clave de Vigenère* es cualquier elemento K de $exp(A)^*$. La función de cifrado sobre el alfabeto A de n símbolos según la clave K es $E_K\colon exp(A)^* \longrightarrow exp(A)^*$ definida como sigue:

$$E_K(s) = \langle f^{-1}((f(s_j) + f(K_j^{len(s)})) \text{ mód } n)\rangle_j$$

donde $s = \langle s_j\rangle_j$, $K^{len(s)}$ es la yuxtaposición de la expresión K consigo misma y eventual truncamiento final hasta conseguir así una expresión con la misma longitud que s. La función de descifrado sobre ese alfabeto y clave es $D_K\colon exp(A)^* \longrightarrow exp(A)^*$ definida por:

$$D_K(s) = \langle f^{-1}((f(s_j) - f(K_j^{len(s)})) \text{ mód } n)\rangle_j$$

Teorema 2.2.1. *Sea $n \in \omega^*$, A un alfabeto de n símbolos y K una clave de Vigenère. Entonces:*

$$D_K \circ E_K = 1_{exp(A)^*}$$

Demostración. Sea $s = \langle s_j \rangle_j$ una expresión de $exp(A)^*$. Entonces:

$$\begin{aligned}
D_K(E_K(s)) &= D_K(\langle f^{-1}((f(s_j) + f(K_j^{len(s)})) \bmod n)\rangle_j) \\
&= \langle f^{-1}((f(f^{-1}((f(s_j) + f(K_j^{len(s)})) \bmod n)) - f(K_j^{len(s)})) \bmod n)\rangle_j \\
&= \langle f^{-1}((((f(s_j) + f(K_j^{len(s)})) \bmod n) - f(K_j^{len(s)})) \bmod n)\rangle_j \\
&= \langle f^{-1}(f(s_j) \bmod n)\rangle_j \\
&= \langle f^{-1}(f(s_j))\rangle_j \\
&= \langle s_j \rangle_j \\
&= s
\end{aligned}$$

□

Teorema 2.2.2. *Sea* $n \in \omega^*$, A *un alfabeto de* n *símbolos y* $K = \langle x_j \rangle_j$ *una clave de Vigenère. Entonces:*

$$D_K = E_{K'}$$

dónde $K' = \langle (-x_j) \bmod n \rangle_j$

El sistema sagemath ofrece el siguiente diálogo:

```
sage: S = AlphabeticStrings()
sage: E = VigenereCryptosystem(S,14)
sage: E
Vigenere cryptosystem on Free alphabetic string monoid on A-Z of period 14
sage: K = S('ABCDEFGHIJKLMN')
sage: K
ABCDEFGHIJKLMN
sage: e = E(K)
sage: e
Cipher on Free alphabetic string monoid on A-Z
sage: e(S("THECATINTHEHAT"))
TIGFEYOUBQOSMG
sage:
sage: V = VigenereCryptosystem(AlphabeticStrings(), 24)
sage: K = V.random_key()
sage: M = V.encoding("Jack and Jill went up the hill.")
sage: V.deciphering(K, V.enciphering(K, M)) == M
True
sage:
sage: A = AlphabeticStrings()
```

```
sage: V = VigenereCryptosystem(A, 24)
sage: M = "Jack and Jill went up the hill."
sage: V.encoding(M) == A.encoding(M)
True
sage:
sage: S = AlphabeticStrings()
sage: E = VigenereCryptosystem(S,14)
sage: K = E.random_key()
sage: L = E.inverse_key(K)
sage: M = S("THECATINTHEHAT")
sage: e = E(K)
sage: c = E(L)
sage: c(e(M))
THECATINTHEHAT
sage:
sage: A = AlphabeticStrings()
sage: V = VigenereCryptosystem(A, 14)
sage: M = "THECATINTHEHAT"
sage: K = V.random_key()
sage: Ki = V.inverse_key(K)
sage: e = V(K)
sage: d = V(Ki)
sage: d(e(A(M))) == A(M)
True
```

La idea detrás del cifrado Vigenère, como todos los demás cifrados polialfabéticos, es disfrazar la frecuencia de las letras de texto llano para interferir con una aplicación directa del análisis de frecuencia. Por ejemplo, si P es la letra más frecuente en un texto cifrado cuyo texto en claro está en inglés, se podría sospechar que P corresponde a E, ya que E es la letra que se usa con más frecuencia en inglés. Sin embargo, al utilizar el cifrado Vigenère, E puede cifrarse como letras de texto cifrado diferentes en diferentes puntos del mensaje, lo que anula el simple análisis de frecuencia.

La principal debilidad del cifrado Vigenère es la estructura de su clave. Si un criptoanalista adivina correctamente la longitud de la clave, el texto cifrado puede tratarse como cifras entrelazadas de César,[3] que pueden romperse fácilmente de forma individual. El *examen de Friedrich Kasiski* y la *prueba de Friedman* pueden ayudar a determinar la longitud de la clave y poner en claro el mensaje.

[3] Aquí llamamos cifra de César a cualquier criptosistema afín de decimación igual a 1.

Examen de Kasiski

En 1863, un veterano del ejército prusiano, el mayor *Friedrich Wilhelm Kasiski*, da el primer paso hacia el criptoanálisis de os cifrados polialfabéticos dentro de un libro de 95 páginas titulado "Die Geheimschriften und die Dechiffrirkunst" ("La escritura secreta y el arte del desciframiento"). Este primer paso no es sino encontrar la longitud de la clave empleada. Como bien observa Kasiski, si dos fragamentos iguales de texto en claro distan uno del otro un múltiplo de la longitud de la clave, entonces sus transformados del texto cifrado también son idénticos. Así pues, si en un criptograma hay bloques repetidos entonces la longitud de la clave podría dividir al máximo común divisor de las distancias entre fragmentos iguales; esto podría llevarnos a su hallazgo.

Aunque Kasiski fue el primero en publicar un relato del ataque, está claro que otros lo habían sabido antes. Un excéntrico científico inglés que invirtió la mayor parte de su tiempo en empeños como: determinar los climas del pasado estudiando la anchura de los anillos de los troncos de los árboles o diseñando una máquina calculadora que en cierto modo podía ser programable. Se trata de Charles Babbage que en 1854, fue incitado a romper el cifrado de Vigenère cuando John Hall Brock Thwaites envió un "nuevo" cifrado al Journal of the Society of the Arts. Cuando Babbage mostró que el cifrado de Thwaites era esencialmente otra recreación del cifrado de Vigenère, Thwaites presentó un desafío a Babbage: dado un texto original (de La tempestad de Shakespeare: acto 1, escena 2) y su versión cifrada, debía encontrar la palabras clave que Thwaites había utilizado para cifrar el texto original. Babbage pronto encontró las palabras clave: "dosz çombinados". Luego, Babbage cifró el mismo pasaje de Shakespeare usando diferentes palabras clave y desafió a Thwaites a encontrar las palabras clave de Babbage. Babbage nunca explicó el método que utilizó. Los estudios de las notas de Babbage revelan que había utilizado el método publicado más tarde por Kasiski y sugieren que había estado utilizando el método ya en 1846.

Como decíamos el examen de Kasiski, también llamado prueba de Kasiski, aprovecha el hecho de que las palabras repetidas, por casualidad, a veces se cifran con las mismas letras clave, lo que lleva a grupos repetidos en el texto cifrado. Por ejemplo, considere el siguiente cifrado utilizando la palabra clave `ABCD`:

clave:	`ABCDABCDABCDABCDABCDABCDABCD`
Texto llano:	`CRYPTOISSHORTFORCRYPTOGRAPHY`
texto cifrado:	`CSASTPKVSIQUTGQUCSASTPIUAQJB`

Hay una repetición fácil de notar en el texto cifrado, por lo que la prueba de Kasiski será efectiva.

La distancia entre las repeticiones de `CSASTP` es 16. Si se supone que los segmentos

repetidos representan los mismos segmentos de texto sin formato, eso implica que la clave tiene 16, 8, 4, 2 ó 1 caracteres de longitud. (Todos los factores de la distancia son longitudes de clave posibles; una clave de longitud uno es simplemente un cifrado César simple, y su criptoanálisis es mucho más fácil). Dado que las longitudes de clave 2 y 1 son poco realistas, es necesario probar solo las longitudes 16, 8 o 4. Los mensajes más largos hacen que la prueba sea más precisa porque generalmente contienen más segmentos de texto cifrado repetidos. El siguiente texto cifrado tiene dos segmentos que se repiten:

texto cifrado: `VHVSSPQUCEMRVBVBBBVHVSURQGIBDUGRNICJQUCERVUAXSSR`

La distancia entre las repeticiones de `VHVS` es 18 y entre las de `QUCE` es de 30 caractéres. Eso significa que la longitud de la clave podría ser un divisor de (18,30)=6, es decir: 6, 3, 2 ó 1. Decartamos 3, 2, y 1 por corresponder a claves demasiado cortas para ser reales y nos quedamos con 6.

Una explicación de esto y otra forma de entenderlo sería el siguiente razonamiento. La distancia entre las repeticiones de `VHVS` es 18. Si se asume que los segmentos repetidos representan los mismos segmentos de texto plano, eso implica que la clave tiene 18, 9, 6, 3, 2 ó 1 carácter. La distancia entre las repeticiones de `QUCE` es de 30 caracteres. Eso significa que la longitud de la clave puede ser de 30, 15, 10, 6, 5, 3, 2 o 1 carácter. Al tomar la intersección de esos conjuntos, se podría concluir con seguridad que la longitud de clave más probable es 6, ya que 3, 2 y 1 son irrealmente cortos.

No obstante el problema de encontrar la longitud de la clave se resuelve de un modo definitivo mediante el llamado *índice de coincidencia* de un texto, concepto introducido por el gran criptoanalista del ejército norteamericano *William Friedman* en la década de los años veinte del siglo 20. Se define como la probabilidad de que dos de sus letras elegidas al azar coincidan.

Prueba de Friedman

La prueba de Friedman (a veces conocida como prueba kappa) fue inventada durante la década de 1920 por William F. Friedman, quien utilizó el índice de coincidencia, que mide la irregularidad de las frecuencias de las letras cifradas para romper el cifrado. Al conocer la probabilidad κ_p de que dos letras del idioma de origen elegidas aleatoriamente sean iguales (alrededor de 0.0685 para inglés "monocase", 0.0755 para el español, 0.0793 para el francés y 0.0736 para el alemán) y la probabilidad de una coincidencia para una selección aleatoria uniforme del alfabeto κ_r ($1/26 = 0.0385$ para inglés), la longitud de la clave se puede estimar de la siguiente manera:

$$\frac{\kappa_p - \kappa_r}{\kappa_o - \kappa_r}$$

de la tasa de coincidencia observada

$$\kappa_o = \frac{\sum_{i=1}^{c} n_i(n_i - 1)}{N(N-1)}$$

en la que c es el tamaño del alfabeto (26 para inglés), N es la longitud del texto y n_1 a n_c son las frecuencias de letras de texto cifrado observadas, como números enteros.

Sin embargo, eso es sólo una aproximación; su precisión aumenta con el tamaño del texto. En la práctica, sería necesario probar varias longitudes de clave cercanas a la estimación. Un mejor enfoque para los cifrados de clave repetida es copiar el texto cifrado en filas de una matriz con tantas columnas como una longitud de clave supuesta y luego calcular el índice promedio de coincidencia con cada columna considerada por separado. Cuando se hace eso para cada longitud de clave posible, el I.C. luego corresponde a la longitud de clave más probable. Estas pruebas pueden complementarse con información del examen de Kasiski.

Análisis de Frecuencias

Una vez que se conoce la longitud de la clave, el texto cifrado puede reescribirse en esa cantidad de columnas, y cada columna corresponde a una sola letra de la clave. Cada columna consta de texto sin formato que ha sido cifrado por un único cifrado Caesar. La clave César (shift) es sólo la letra de la clave Vigenère que se usó para esa columna. Utilizando métodos similares a los utilizados para descifrar el cifrado César, se pueden descubrir las letras del texto cifrado.

Una mejora en el examen de Kasiski, conocida como *método de Kerckhoffs*, hace coincidir las frecuencias de letras de cada columna con las frecuencias de texto plano desplazadas para descubrir la letra clave (cambio César) para esa columna. Una vez que se conocen todas las letras de la clave, todo lo que el criptoanalista tiene que hacer es descifrar el texto cifrado y revelar el texto sin formato. El método de Kerckhoffs no es aplicable si la tabla de Vigenère se ha codificado, en lugar de usar secuencias alfabéticas normales, pero el examen de Kasiski y las pruebas de coincidencia aún se pueden usar para determinar la longitud de la clave.

Eliminación de Clave

El cifrado Vigenère, con alfabetos normales, utiliza esencialmente aritmética módulo, que es conmutativa. Por lo tanto, si se conoce (o se adivina) la longitud de la clave, restar el texto cifrado de sí mismo, compensado por la longitud m de la clave, producirá el texto llano restado de sí mismo también compensado por la longitud de la clave. Si se conoce o se puede adivinar alguna "palabra probable" en el texto llano se puede reconocer su auto-sustracción, lo que permite recuperar la clave restando el texto llano conocido del texto cifrado. La eliminación de claves es especialmente útil contra mensajes cortos. Por ejemplo, usando LION como clave a continuación:

texto llano:	`THEQUICKBROWNFOXJUMPSOVERTHELAZYDOG`
clave:	`LIONLIONLIONLIONLIONLIONLIONLIONLIO`
texto cifrado:	`EPSDFQQXMZCJYNCKUCACDWJRCBVRWINLOWU`

Luego reste el texto cifrado de sí mismo con un cambio de la longitud de clave 4 para LION.

texto cifrado (original)	`EPSDFQQXMZCJYNCKUCACDWJRCBVRWINLOWU`
texto cifrado (desplazado):	`FQQXMZCJYNCKUCACDWJRCBVRWINLOWU____`
resultado (diferencia):	`ZZCGTROOOMAZELCIRGRLBVOAGTIGIMT~~LOWU~~`

Lo que es casi equivalente a restar el texto llano de sí mismo mediante el mismo desplazamiento.

texto llano (original):	`THEQUICKBROWNFOXJUMPSOVERTHELAZYDOG`
texto llano (desplazado):	`UICKBROWNFOXJUMPSOVERTHELAZYDOG____`
resultado (diferencia):	`ZZCGTROOOMAZELCIRGRLBVOAGTIGIMT~~YDOG~~`

Lo que se representa algebraicamente para $1 \leq i \leq n - m$ como:

$$
\begin{aligned}
(C_i - C_{i+m}) \bmod l &= (E_K(M_i) - E_K(M_{i+m})) \bmod l \\
&= ((M_i + K_{i \bmod m}) \bmod l - (M_{i+m} + K_{(i+m) \bmod m}) \bmod l) \bmod l \\
&= ((M_i + K_{i \bmod m}) - (M_{i+m} + K_{(i+m) \bmod m})) \bmod l \\
&= (M_i + K_{i \bmod m} - M_{i+m} - K_{(i+m) \bmod m}) \bmod l \\
&= (M_i - M_{i+m} + K_{i \bmod m} - K_{(i+m) \bmod m}) \bmod l \\
&= (M_i - M_{i+m} + K_{i \bmod m} - K_{i \bmod m}) \bmod l \\
&= (M_i - M_{i+m}) \bmod l
\end{aligned}
$$

En este ejemplo, las palabras BROWNFOX son conocidas.

texto cifrado (original)	BROWNFOX
texto cifrado (desplazado):	NFOX____
resultado (diferencia)	OMAZ~~NFOX~~

Este resultado OMAZ se corresponde con las letras 9 a 12 en el resultado de los ejemplos más grandes anteriores. Se verifica la sección conocida y su ubicación.

Reste BROW de ese rango del texto cifrado.

texto cifrado:	EPSDFQQXMZCJYNCKUCACDWJRCBVRWINLOWU
texto llano:	_____BROW_____
clave:	~~EPSDFQQX~~LION~~YNCKUCACDWJRCBVRWINLOWU~~

Variante

La variante running key (clave en ejecución) del cifrado Vigenère también se consideró irrompible en algún momento. Esta versión utiliza como clave un bloque de texto tan largo como el texto sin formato. Dado que la clave es tan larga como el mensaje, las pruebas de Friedman y Kasiski ya no funcionan, ya que la clave no se repite.

Si se utilizan varias claves, la longitud efectiva de la clave es el mínimo común múltiplo de las longitudes de las claves individuales. Por ejemplo, utilizando las dos claves GO y CAT, cuyas longitudes son 2 y 3, se obtiene una longitud de clave efectiva de 6 (el mínimo común múltiplo de 2 y 3). Esto puede entenderse como el punto en el que ambas teclas se alinean.

texto llano	ATTACKATDAWN
clave 1:	GOGOGOGOGOGO
clave 2:	CATCATCATCAT
texto cifrado:	IHSQIRIHCQCU

Cifrar dos veces, primero con la clave GO y luego con la clave CAT es lo mismo que cifrar una vez con una clave producida al cifrar una clave con la otra.

texto llano	GOGOGO
clave:	CATCAT
texto cifrado:	IOZQGH

Esto se demuestra cifrando ATTACKATDAWN con IOZQGH, para producir el mismo texto cifrado que en el ejemplo original.

texto llano	ATTACKATDAWN
clave:	IOZQGHIOZQGH
texto cifrado	IHSQIRIHCQCU

Si las longitudes de las claves son números primos relativos, la longitud efectiva de las claves aumenta exponencialmente a medida que aumentan las longitudes de las claves individuales. Esto es especialmente cierto si cada longitud de clave es primo individualmente. Por ejemplo, la longitud efectiva de las claves 2, 3 y 5 caracteres es 30, pero la de las claves de 7, 11 y 13 caracteres es 1001. Si esta longitud de clave efectiva es más larga que el texto cifrado, logra la misma inmunidad a las pruebas de Friedman y Kasiski que la variante de clave en ejecución.

Si uno usa una clave que es verdaderamente aleatoria, es al menos tan larga como el mensaje cifrado y se usa solo una vez, el cifrado de Vigenère es teóricamente irrompible. Sin embargo, en ese caso la clave, no el cifrado, proporciona fuerza criptográfica y tales sistemas se denominan de manera adecuada en conjunto como sistemas one-time pad (relleno único), independientemente de los cifrados empleados.

Vigenère en realidad inventó un cifrado más fuerte, un cifrado de autokey cipher (clave automática). En cambio, el nombre "cifrado Vigenère" se asoció con un cifrado polialfabético más simple. De hecho, los dos cifrados se confundían a menudo y en ocasiones ambos se llamaban *le chiffre indéchiffrable*. Babbage realmente rompió el cifrado de clave automática más fuerte, pero a Kasiski generalmente se le atribuye la primera solución publicada para los cifrados polialfabéticos de clave fija.

Una variante simple es cifrar utilizando el método de descifrado de Vigenère y descifrar utilizando el cifrado de Vigenère. Ese método a veces se denomina *Variante Beaufort*. Es diferente del cifrado de Beaufort, creado por Francis Beaufort, que es similar al Vigenère pero utilizando un mecanismo de cifrado y un cuadro ligeramente modificado. El cifrado de Beaufort es un cifrado recíproco.

A pesar de la aparente fuerza del cifrado Vigenère, nunca llegó a ser ampliamente utilizado en toda Europa. El cifrado de Gronsfeld es una variante creada por el Conde Gronsfeld (Josse Maximilaan van Gronsveld né van Bronckhorst); es idéntico al cifrado de Vigenère, excepto que utiliza sólo 10 (alfabetos de cifrado diferentes¿?) caractéres diferentes en el alfabeto para la clave correspondientes a los dígitos del 0 al 9. Una clave de Gronsfeld de 0123 es lo mismo que una clave de Vigenère de ABCD. El cifrado de Gronsfeld se fortalece porque su clave no es una palabra, pero se debilita porque tiene solo 10 caracteres para formar la clave (alfabetos de cifrado). Es el cifrado de Gronsfeld el que se utilizó ampliamente en Alemania y Europa, a pesar de sus debilidades.

Ejercicio 2.2.1. Haga lo siguiente:

1. Idee una extensión del criptosistema de Vigenère mediante el cifrado afín y valore su fortaleza.

2. Implemente las herramientas necesarias para poder llevar a cabo el examen de Kasiski.

3. Implemente las herramientas necesarias para poder llevar a cabo la prueba de Friedman.

Para llevar a cabo este ejercicio puede consultar la teoría y el ejemplo de [7] entre las páginas 218 y 227 así como el ejemplo de [12] entre las páginas 59 y 70.

2.3. Criptosistema de Hill

El *criptosistema de Hill* es una cifra polialfabética. Inventado por Lester S. Hill en 1929, fue el primer sistema criptográfico polialfabético práctico para trabajar con más de tres símbolos simultaneamente. Desconocemos un uso histórico señalado, si bien es cierto que es de gran interés como mecanismo desestructurador complementario a otros criptosistemas.

Definición 2.3.1. Sea $n \in \omega^*$ y A un alfabeto de n símbolos. Una *clave de Hill* es cualquier par $\langle m, T\rangle$, donde $m \in \omega^*$ y $T \in M_m(Z_n)$ es regular. La función de cifrado sobre el alfabeto A de n símbolos según la llave $\langle m, T\rangle$ es la función $E_{\langle m,T\rangle} : exp(A)^* \longrightarrow exp(A)^*$ definida como sigue:

$$E_{\langle m,T\rangle}(s) = \biguplus_{k=0}^{\operatorname{len}(s') \operatorname{div} m} f^{-1}(T(\langle f(s'_{k+j})\rangle_{j=0}^{m-1})^t)^t$$

donde: los resultados de multiplicar matrices por vectores en $\mathbb{Z}_n$ se consideran de componentes reducidas módulo n, en un abuso de lenguaje $f^{-1}(\langle x_i\rangle)$ es $\langle f^{-1}(x_i)\rangle$,

$$s' = \begin{cases} s, & \text{si } \operatorname{len}(s) \equiv 0 \pmod m \\ s \uplus \langle w\rangle_{j=0}^{m-(\operatorname{len}(s) \bmod m)-1}, & \text{en otro caso} \end{cases}$$

y $\uplus$ representa la concatenación de secuencias.[4]

Observación 2.3.1. Obsérvese que el cifrado de Hill para claves con $m = 1$ es un caso particular del cifrado afín.

[4] La elección de w es por mera operatividad, al ser una letra poco frecuente. Según la lengua en cuestión se puede convenir completar con esa letra u otra.

Teorema 2.3.1. *Sea* $n \in \omega^*$, A *un alfabeto de* n *símbolo y* $\langle m, T \rangle$ *una clave de Hill. Entonces:* $D_{\langle m,T \rangle} = E_{\langle m,T^{-1} \rangle}$

El sistema sagemath aporta una implementación de criptosistema de Hill. Como ejemplo sugiere el siguiente diálogo:

```
sage: S = AlphabeticStrings()
sage: E = HillCryptosystem(S,3)
sage: E
Hill cryptosystem on Free alphabetic string monoid on A-Z of block length 3
sage: R = IntegerModRing(26)
sage: M = MatrixSpace(R,3,3)
sage: A = M([[1,0,1],[0,1,1],[2,2,3]])
sage: A
[1 0 1]
[0 1 1]
[2 2 3]
sage: e = E(A)
sage: e
Hill cipher on Free alphabetic string monoid on A-Z of block length 3
sage: e(S("LAMAISONBLANCHE"))
JYVKSKQPELAYKPV

sage: A = AlphabeticStrings()
sage: n = randint(1, A.ngens() - 1)
sage: H = HillCryptosystem(A, n)
sage: H.block_length() == n
True

sage: H = HillCryptosystem(AlphabeticStrings(), 3)
sage: K = H.random_key()
sage: M = H.encoding("Good day, mate! How ya going?")
sage: H.deciphering(K, H.enciphering(K, M)) == M
True


sage: M = "The matrix cipher by Lester S. Hill."
sage: A = AlphabeticStrings()
sage: H = HillCryptosystem(A, 7)
sage: H.encoding(M) == A.encoding(M)
True
```

```
sage: S = AlphabeticStrings()
sage: E = HillCryptosystem(S,3)
sage: A = E.random_key()
sage: B = E.inverse_key(A)
sage: M = S("LAMAISONBLANCHE")
sage: e = E(A)
sage: c = E(B)
sage: c(e(M))
LAMAISONBLANCHE

sage: A = AlphabeticStrings()
sage: n = 3
sage: H = HillCryptosystem(A, n)
sage: K = H.random_key()
sage: Ki = H.inverse_key(K)
sage: M = "LAMAISONBLANCHE"
sage: e = H(K)
sage: d = H(Ki)
sage: d(e(A(M))) == A(M)
True
```

Seguridad

El cifrado básico de Hill es vulnerable a un ataque de texto plano conocido porque es completamente lineal. Un oponente que intercepta n pares de caracteres de texto llano/texto cifrado puede establecer un sistema lineal que (normalmente) se puede resolver fácilmente; si sucede que este sistema es indeterminado, solo es necesario agregar algunos pares más de texto llano/texto cifrado. El cálculo de esta solución mediante algoritmos estándar de álgebra lineal lleva muy poco tiempo. *Konheim* en [9] describe con detalle este ataque.

Si bien la multiplicación de matrices por sí sola no da como resultado un cifrado seguro, sigue siendo un paso útil cuando se combina con otras operaciones no lineales, porque la multiplicación de matrices puede proporcionar difusión. Por ejemplo, una matriz elegida apropiadamente puede garantizar que pequeñas diferencias antes de la multiplicación de matrices resultarán en grandes diferencias después de la multiplicación de matrices. De hecho, algunos cifrados modernos utilizan un paso de multiplicación de matrices para proporcionar difusión. Por ejemplo, el paso `MixColumns` en AES es una multiplicación de matrices. La función g en `Twofish` es una combinación de S-boxes no lineales con

una multiplicación de matrices (MDS) cuidadosamente elegida.

Teorema 2.3.2. *Sea* p *un número primo,* $m \in \omega^*$ *y sea* $g(p,m)$ *el número de matrices regulares de* $M_m(\mathbb{Z}_p)$*. Entonces:*

$$g(p,m) = \prod_{j=0}^{m-1}(p^m - p^j)$$

Demostración. Para una demostración cfr. número de matrices invertibles sobre cuerpos finitos. □

Ejercicio 2.3.1. Haga los siguientes ejercicios:

1. Implemente el criptosistema de Hill hasta donde le sea posible.
2. Imagine una extensión del criptosistema de Hill inspirada en el cifrado afín.

Capítulo 3

El Criptosistema DES

3.1. Introducción

Los *criptosistemas iterativos* forman una familia de funciones criptográficas fuertes basadas en iterar un número de veces adecuado una función débil. A las iteraciones se las denomina *rondas* y el criptosistema es denominado "un criptosistema de n-rondas". La "función ronda" es una función de la salida aportada por la ronda previa y de una *subllave*. La función de ronda depende usualmente de tablas —también llamadas *sustituciones* o *S-cajas*—, permutaciones de bits, operaciones aritméticas y la *o exclusiva* —denotada por $\oplus$ o XOR y que corresponde a la función booleana de dos variables $x \oplus y = (x \wedge \neg y) \vee (\neg x \wedge y)$—. Como es habitual el algoritmo de cifrado no se mantiene secreto, confiando en que sin conocer la llave, que si será secreta, el algoritmo tiene un elevadísimo índice de seguridad.

Hasta principio de los 70, la investigación criptografía no militar se realizaba al azar. Casi ningún artículo fue publicado al respecto, aunque todo el mundo sabía que los militares utilizaban códigos especiales para comunicarse. Casi nadie tenía conocimientos sobre esa sofisticada ciencia que empezó a llamarse *Criptografía*. La *National Security Agence* (*NSA*) estadounidense tenía un amplio expediente sobre el tema sin que llegara admitirlo públicamente y mucho menos a darlo a conocer. Varias pequeñas compañías confeccionaron y empezaron a vender paquetes criptográficos. Sin embargo, no eran compatibles entre sí y no se sabía con certeza la fortaleza de los mismos.

El *National Bureau of Standards* (*NBS*) estadounidense —ahora el *National Institute of Standards and Technology*, (*NIST*)— abrió un programa tendente a adoptar un sistema estandar de protección de datos informáticos (almacenados o transmitidos) y de comunicaciones. El paquete estandar debería tener a los ojos del *NBS* las siguientes

características, propuestas el 15 de mayo de 1973 en el *Registro Federal*:

- El algoritmo debería proporcionar un elevado grado de seguridad.
- El algoritmo debe estar completamente especificado y ser fácil de entender.
- La seguridad del algoritmo dependerá exclusivamente de la llave y no de la secrecidad del mismo.
- El algoritmo deberá estar disponible para todos los usuarios.
- El algoritmo deberá ser adaptable para su utilización en diversas aplicaciones.
- El algoritmo deberá ser fácilmente implementado en sistemas electrónicos.
- El algoritmo deberá ser eficiente.
- El algoritmo deberá se exportable.

La respuesta a la propuesta fue extensa pero insatisfactoria. Así pues, el 27 de agosto de 1974 fue emitida una segunda convocatoria.

La casa *IBM* propuso la utilización de un sistema de cifrado iterativo que fue denominado *Lucifer*. El diseño de *Lucifer* fue realizado por *IBM* como medida para proteger la seguridad de los datos en sus productos. *DES* es una versión mejorada de *Lucifer* que fue desarrollada también por *IBM*. El *NBS* estadounidense lo adoptó el 23 de noviembre de 1976 como criptosistema estandar para la protección en las comunicaciones de material gubernamental no clasificado. El sistema *DES* fue validado, con más o menos reservas, hasta la actualidad. La reticencias, sobre todo en la revisión del 87, no venían de que el sistema hubiera sido roto, sino de que los medios informáticos crecientes hacían previsible a corto plazo el ataque exitoso. El 19 de mayo de 2005 fue definitivamente retirado, permitiendo el cifrado de material gubernamental sensible hasta 2030 bajo la modalidad "Triple DES". De todas formas, el 26 de mayo de 2002 el criptosistema DES había sido sobrepasado en uso por el criptosistema Advanced Encryption Standard (AES).

3.2. Breve Descripción de DES

DES es un algoritmo de cifrado por bloques. Cifra bloques de 64 bits de texto llano y no altera el tamaño del bloque en el resultado de su cifrado. El algoritmo de cifrado coincide con el de descifrado, salvo que en ambos procesos las listas de las llaves son distintas.

Cada llave es una lista de 56 bits, aunque se da como una lista de 64 bits. Lo que ocurre en realidad es que los bits que usan las posiciones: 8, 16, 24, 32, 40, 48, 56 y 64 son utilizadas para el control de paridad y, salvo esto, ignoradas en el proceso.

No hay restricciones para las llaves aparte de su tamaño y pueden ser cambiadas en cualquier momento. Hay llaves que se consideran débiles, pero pueden ser fácilmente detectadas y despreciadas.

El algoritmo en sí no es otra cosa que una delicada combinación de dos técnicas habituales en el cifrado:

- *Confusión.* Oscurece la relación entre texto cifrado y texto cifrado. Hará que se sienta frustado el criptoanalista que intente abordar la ruptura del criptosistema sin más herramientas que la búsqueda de redundancias o patrones estadísticos. La sustitución, el *método de Cesar* por ejemplo, representa un tibio acercamiento a la confusión. Sin embargo, el concepto moderno de confusión es más sofisticado. La *máquina ENIGMA* modifica su forma —que no su método— de sustituir cada vez que sustituye, aunque fue rota en la época inmediatamente anterior a los ordenadores.
- *Difusión.* Disipa a lo largo del texto cifrado la redundancia —inherente al lenguaje natural— que pueda haber en cada muestra de texto llano. El criptoanalista tendrá que perder un tiempo precioso intentando recomponer la redundancia, inferir un método general para hacerlo y situar adecuadamente los resultados del proceso. La forma más simple de producir difusión es mediante el uso de *permutaciones* . La difusión por si misma es débil. Los modernos cifradores combinan la confusión con la difusión.

Repite 16 veces un mismo proceso al que se le denomina *ronda* y las operaciones que emplean no van más allá de la aritmética estandar y de las operaciones del Algebra de Boole. En los años 70 se implementó en hardware mientras que las implementaciones en software, que ahora han mejorado, eran bastante toscas. El esquema repetitivo del algoritmo lo hacen muy adecuado para usarlo en un chip de propósito específico.

3.3. El Esquema de DES

La entrada del algoritmo es un bloque B_{input} de 64 bits de texto llano y la salida otro del mismo tamaño, B_{output}, que constituirá una parte del texto cifrado. El proceso que se sigue a *grosso modo* —después explicaremos los detalles más detenidamente— es el presentado en la Figura 3.1.

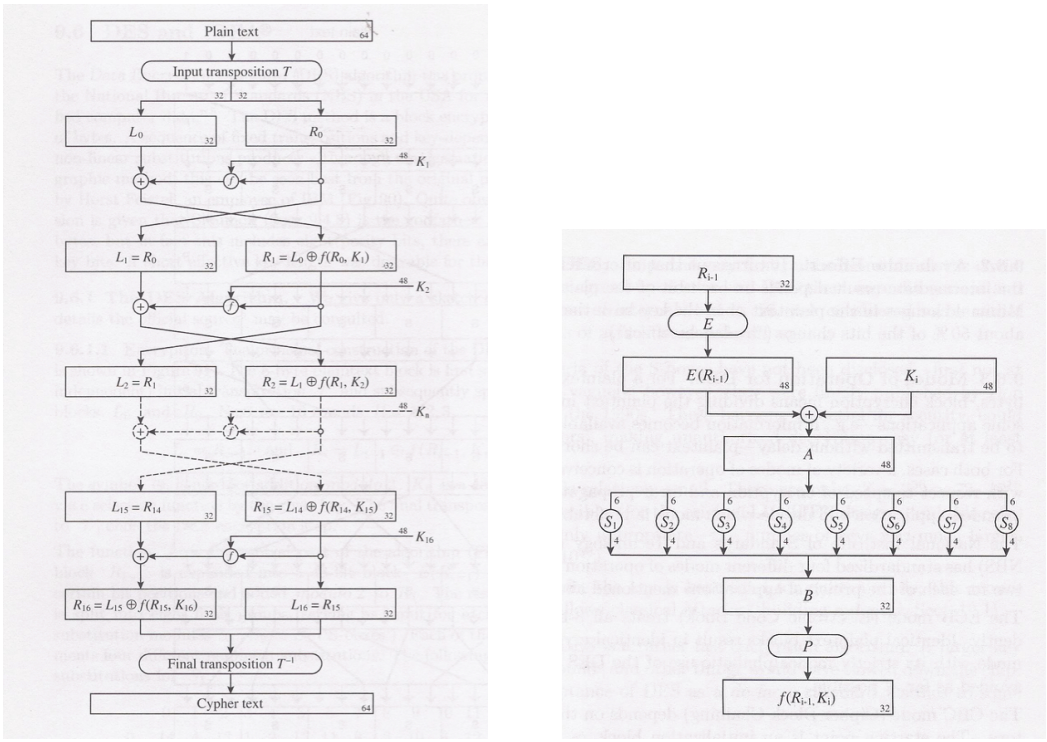
input: Una palabra de 64 bits $B_{input} = \langle l_1, \ldots, l_{64} \rangle$ de texto llano.
output: Una palabra de 64 bits $B_{output} = \langle c_1, \ldots, c_{64} \rangle$ de texto cifrado.
requisito: Palabras de 64 bits cada una K_i, $1 \leq i \leq 16$.
procedure $DES(B_{input}, K_1, \ldots, K_{16}; B_{output})$
    begin
01.) **for** $i = 1$ **to** 64 **do**
02.) $B[IP(i)] \leftarrow B_{input}[i]$
    od
03.) $L \leftarrow \pi_1\big(Divide(B; L, R)\big)$
    "Divide" divide la palabra B de 64 bits en dos de 32 bits
    (los 32 primeros y el resto) y presenta un par $\langle L, R \rangle$
04.) $L \leftarrow \pi_2\big(Divide(B; L, R)\big)$
05.) $i \leftarrow 1$
06.) **while** $i \leq 15$ **do**
07.) $M \leftarrow L$
07.) $L \leftarrow R$
08.) $R \leftarrow M \oplus f(L, K_i)$
      f es una func. que se define posteriormente
09.) $i \leftarrow i + 1$
    od
10.) $I \leftarrow L \oplus f(R, K_{16})$
11.) $Unir(I, R; Q)$
    "Unir" yuxtapone la 32-upla R a la I y produce la 64-upla Q.
12.) **for** $i = 1$ **to** 64 **do**
13.) $B_{output}[IP^{-1}(i)] \leftarrow Q[i]$
    od
14.) **return** B_{output}
    end

Figura 3.1: Procedimiento DES para cifrar un bloque de texto llano de 64 bits según el algoritmo "*Data Encryption Standard*".

La Permutación Inicial

La presencia de la permutación inicial IP y de su inversa al final del algoritmo no afectan a la seguridad de *DES*. Muchas implementaciones en software de *DES* han descartado aplicar IP y ello no las hace más débiles, sin embargo no se atienen al estandar y por tanto no puede llamarsele *DES*. La permutación estandar es la presentada en la Figura 3.3. Para entender esta lista[1] hay que leerla de izquierda a derecha y de arriba abajo. Indica que la i-ésima posición de la lista resultante será ocupada por el bit de la lista original cuyo número de orden figura en la entrada i de la tabla. Por ejemplo, si tomamos la Tabla 3.3 debemos entender que el bit de la posición 1 en la lista de salida es el bit que en la lista de entrada ocupa la posición 58, el de la posición 2 es el de la posición 50, etc. Si consideramos la Tabla 3.3 como una biyección IP de $\mathbb{Z}_{64}$ en sí mismo, hay que entender que dicha biyección asociaría, por ejemplo, el 1 al 40, el 2 al 8, el 3 al 48 y el 58 al 1, o sea, $\mathrm{IP}(1) = 40$, $\mathrm{IP}(2) = 8$, $\mathrm{IP}(3) = 48$ y $\mathrm{IP}(58) = 1$.

[1] Este comentario y los siguientes valen para el resto de situaciones análogas.



(a) Pasos de cifrado con DES (b) La función f

Figura 3.2: Esquema de detalles básicos del cifrado con DES

La Transformación de las Llaves

Inicialmente la llave de 64 bits es reducida a una llave de 56 bits ignorando los bits de paridad que son los que ocupan posiciones múltiplo de 8 y sometiendo el resto a una permutación. Una vez que que se ha extraido la llave de 56 bits se procede a generar una de 48 bits para cada ronda del algoritmo que llamaremos K_i, $1 \leq i \leq 16$. El proceso que se sigue es el siguiente:

1. En la llave inicial de 64 bits se descartan los bits que ocupan posiciones múltiplo de 8 y el resto se reordenan según la permutación *PC1* de la Figura 3.4.
2. La llave de 56 bits es dividida en dos paquetes de 28 bits mediante la función $P(K; C, D)$ que genera el par $\langle C, D\rangle$ cuando se le pone a disposición la llave K. Es decir, si $K = \langle k_1, \ldots, k_{28}, k_{29}, \ldots, k_{56}\rangle$, entonces $C = \langle k_1, \ldots, k_{28}\rangle$ y $D = \langle k_{29}, \ldots, k_{56}\rangle$.
3. Los paquetes son desplazados a la izquierda en una o dos posiciones, dependiendo de la ronda. La cuantía del desplazamiento según la ronda, $sl(i)$, está dada en la Tabla 3.5. En cada paquete y en la ronda i-ésima, el bit j-ésimo es situado en la posición $j - sl(i)$ mód 28.
4. Después del desplazamiento se seleccionan 48 bits de los 56. En esta operación, a la vez que se seleccionan ciertos bit, éstos vuelven a ser permutados. La operación es llevada a cabo aplicando la permutación de la Figura 3.6.

58	50	42	34	26	18	10	02	60	52	44	36	28	20	12	04
62	54	46	38	30	22	14	06	64	56	48	40	32	24	16	08
57	49	41	33	25	17	09	01	59	51	43	35	27	19	11	03
61	53	45	37	29	21	13	05	63	55	47	39	31	23	15	07

Figura 3.3: Permutación Inicial IP de *DES*.

57	49	41	33	25	17	09
01	58	50	42	34	26	18
10	02	59	51	43	35	27
19	11	03	60	52	44	36
63	55	47	39	31	23	15
07	62	54	46	38	30	22
14	06	61	53	45	37	29
21	13	05	28	20	12	04

Figura 3.4: Permutación *PC1* para la llave de 64 bits reducida a 56 bits.

Las operaciones descritas anteriormente para la elaboración de las llaves están ordenadas y expuestas en el algoritmo de la Figura 3.7.

La Permutación de Expansión

El objetivo de la operación de expansión es convertir la mitad derecha del dato que tiene 32 bits en una cadena de 48 bits. Esto se lleva a cabo repitiendo bits a la vez que se reordenan. La intención es doble:

- El resultado debe tener la misma longitud que la llave para poder compararla con ella misma mediante la operación $\oplus$.
- Aportar un resultado lo suficientemente largo como para poder ser comprimido en una operación futura, la operación de sustitución.

Ninguna de ellas responde al verdadero interés criptográfico. Permitiendo que un mismo bit sirva para sustituir en dos ocasiones hace que la dependencia de entre resultados y datos se aleatorice. Estamos hablando del *efecto avalancha* que está en la médula de DES. En efecto, en [9] se puede comprobar como pequeñas variaciaciones en la llave o en el texto llano provoca grandes distanciamientos en los resultados.

En la tabla de la Figura 3.8 aparece la permutación EB entorno a la que se vertebra la operación de expansión, operación a la que también se le ha llamado *E-caja*.

Ronda i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Despla. sl(i)	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Figura 3.5: Desplazamiento de la llave por ronda en *DES*.

14	17	11	24	01	05
03	28	15	06	21	10
23	19	12	04	26	08
16	07	27	20	13	02
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Figura 3.6: Permutación *PC2* para la llave.

input: Una palabra K de 64 bits (la llave) e $1 \leq i \leq 16$ (la ronda).
output: Una palabra de 48 bits K_i (la llave de la ronda i).
procedure KEY(i)(K, i; K_i)
    begin
01.) $j \leftarrow 1$
02.) **while** $j \leq 56$ **do**
03.) **if** $8 \mid j$ **then** $j \leftarrow j + 1$
04.) **else do**
05.) $H[PC1(j)] \leftarrow K[j]$
06.) $j \leftarrow j + 1$
      od
    od
07.) $C \leftarrow \pi_1\big(P(H; C, D)\big)$
08.) $D \leftarrow \pi_2\big(P(H; C, D)\big)$
09.) $X \leftarrow C$
10.) **for** $j = 1$ **to** 28 **do**
11.) $X[j - sl(i) \text{ mód } 28] \leftarrow C[j]$
    od
12.) $C \leftarrow X$
13.) **for** $j = 1$ **to** 28 **do**
14.) $X[j - sl(i) \text{ mód } 28] \leftarrow D[j]$
    od
15.) $D \leftarrow X$
16.) $Unir(C, D; H_i)$
    "La operac. Unir yuxtapone la cadena D a la C generando H_i"
17.) $j \leftarrow 1$
18.) $X \leftarrow \{9, 18, 22, 25, 35, 38, 43, 54\}$
19.) **while** $j \leq 56$ **do**
20.) **if** $j \in X$ **then** $j \leftarrow j + 1$
21.) **else do**
22.) $K_i[PC2(j)] \leftarrow H_i[j]$
23.) $j \leftarrow j + 1$
      od
    od
24.) **return** K_i
    end

Figura 3.7: Procedimiento KEY(i) para generar a partir de K la llave K_i de la ronda i-ésima en DES.

Hay que observar que aunque el bloque de salida es más largo que el de entrada no hay más que un bloque de entrada que produzca un bloque concreto de salida. Lo anterior supone afirmar que la operación de extensión es inyectiva.

Las operaciones de:

- Generación de la llave apropiada para la vuelta.
- Expansión de los datos.

culmina en la mezcla de las salidas de ambos procesos mediante la operación $\oplus$. El proceso se sumariza en el esquema de la Figura 3.9 y téngase en cuenta que el mencionado esquema es específico para la tabla PC2 de la Figura 3.8.

Las S-cajas

El resultado de operar mediante $\oplus$ la llave comprimida y el bloque expandido —que es un bloque de 48 bits— se somete a una sustitución. Esta sustitución es compuesta de ocho y cada una de las ocho se lleva a cabo en lo que se denomina una S-caja. La operación no consiste en someter el bloque de 48 bits a 8 rondas consecutivas de permutación, sino que se dividirá en 8 bloques de 6 bits y cada bloque se operará independientemente en una caja.

Una S-caja será representada por una tabla de 4 filas y 16 columnas, de la 0 a la 3 y de la 0 a la 15 respectivamente. Cada entrada de la tabla es un número de cuatro bits, es decir, un número entre 0 y 15. Si $\langle b_1, \ldots, b_6 \rangle$ es el bloque de entrada a la S-caja i-ésima, consideramos $\langle b_1, b_6 \rangle$ y componemos el número $r = b_1 2 + b_6$, que está entre 0 y 3. Dicho número corresponderá a la fila de la S-caja. Con la upla $\langle b_2, b_3, b_4, b_5 \rangle$

32	01	02	03	04	05
04	05	06	07	08	09
08	09	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	01

Figura 3.8: Permutación EB de extensión para la *E-caja*.

input: $1 \leq i \leq 16$, la llave K_i, y un bloque de 48 bits R_{i-1}.
output: Un bloque B_i de 48 bits.
procedure XOR$(K_i, R_{i-1}; B_i)$
begin
01.) **for** $j = 1$ **to** 8 **do**
02.) $H_i[6j - 1 \text{ mód } 48] \leftarrow R_{i-1}[4j \text{ mód } 32]$
03.) $H_i[6j \text{ mód } 48] \leftarrow R_{i-1}[4j + 1 \text{ mód } 32]$
04.) $H_i[6j + 1 \text{ mód } 48] \leftarrow R_{i-1}[4j \text{ mód } 32]$
05.) $H_i[6j + 2 \text{ mód } 48] \leftarrow R_{i-1}[4j + 1 \text{ mód } 32]$
od
06.) **for** $j = 0$ **to** 7 **do**
07.) $H_i[6(j + 1) - 3] \leftarrow R_{i-1}[2 + 4j]$
08.) $H_i[6(j + 1) - 2] \leftarrow R_{i-1}[3 + 4j]$
od
09.) **for** $j = 1$ **to** 48 **do**
10.) $B_i[j] \leftarrow R_{i-1}[j] \oplus H_i[j]$
11.) **return** B_i
end

Figura 3.9: Procedimiento XOR para expandir R_{i-1} y proporcionar la entrada a la S-caja.

obtenemos el número

$$c = b_5 + b_4 2 + b_3 2^2 + b_2 2^3$$

que representará la columna de la S-caja. El valor que en la tabla de la S-caja ocupa la posición $\langle r, c\rangle$ es tomado y expresado en binario. La última operación aportará una cadena de 4 bits, que junto a las 8 restantes —una por S-caja— proporcionará un bloque de 32 bits. En la Figura 3.11 reproducimos las S-cajas del estandar y en la Figura 3.10 esquematizamos el proceso recién descrito.

---

input: Un bloque B de 48 bits.
output: Un bloque S de 32 bits.
procedure SC(B; S)
begin
01.) $U(B; P_1, \ldots, P_8)$
"El procedimiento U divide un bloque de 48 bits en 8 de 6"
02.) **for** $i = 1$ **to** 8 **do**
03.) $L \leftarrow \langle P_i[1], P_i[6]\rangle$
04.) $C \leftarrow \langle P_i[2], P_i[3], P_i[4], P_i[5]\rangle$
05.) $x \leftarrow 2P_i[1] + P_i[6]$
06.) $y \leftarrow 2^3 P_i[2] + 2^2 P_i[3] + 2P_i[4] + P_i[5]$
07.) $s \leftarrow$ S-caja(i, x, y)
08.) $S_i \leftarrow BIN(s)$
od
09.) $S \leftarrow S_1$
09.) **for** $i = 2$ **to** 8 **do**
10.) $S \leftarrow Unir(S, S_i; H_i)$
od
return S
end

Figura 3.10: Procedimiento SC que hace actuar las S-cajas sobre un bloque inicial de 48 bits

La P-caja y la Permutación Final

La última operación que se efectua en la función f es la de aplicar la $P - caja$. Esta operación no es más que aplicar una permutación directa —sin suprimir ni expandir— P al bloque que resulta de aplicar la S-caja. La permutación del estandar es la de la Figura 3.12.

S-caja 1															
14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
00	15	07	04	14	02	13	01	10	06	12	11	09	05	03	08
04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13
S-caja 2															
15	01	08	14	06	11	03	04	09	07	02	13	12	00	05	10
03	13	04	07	15	02	08	14	12	00	01	10	06	09	11	05
00	14	07	11	10	04	13	01	05	08	12	06	09	03	02	15
13	08	10	01	03	15	04	02	11	06	07	12	00	05	14	09
S-caja 3															
10	00	09	14	06	03	15	05	01	13	12	07	11	04	02	08
13	07	00	09	03	04	06	10	02	08	05	14	12	11	15	01
13	06	04	09	08	15	03	00	11	01	02	12	05	10	14	07
01	10	13	00	06	09	08	07	04	15	14	03	11	05	02	12
S-caja 4															
07	13	14	03	00	06	09	10	01	02	08	05	11	12	04	15
13	08	11	05	06	15	00	03	04	07	02	12	01	10	14	09
10	06	09	00	12	11	07	13	15	01	03	14	05	02	08	04
03	15	00	06	10	01	13	08	09	04	05	11	12	07	02	14
S-caja 5															
02	12	04	01	07	10	11	06	08	05	03	15	13	00	14	09
14	11	02	12	04	07	13	01	05	00	15	10	03	09	08	06
04	02	01	11	10	13	07	08	15	09	12	05	06	03	00	14
11	08	12	07	01	14	02	13	06	15	00	09	10	04	05	03
S-caja 6															
12	01	10	15	09	02	06	08	00	13	03	04	04	07	05	11
10	15	04	02	07	12	09	05	06	01	13	14	00	11	03	08
09	14	15	05	02	08	12	03	07	00	04	10	01	13	11	06
04	03	02	12	09	05	15	10	11	14	01	07	06	00	08	13
S-caja 7															
04	11	02	14	15	00	08	13	03	12	09	07	05	10	06	01
13	00	11	07	04	09	01	10	14	03	05	12	02	15	08	06
01	04	11	13	12	03	07	14	10	15	06	08	00	05	09	02
06	11	13	08	01	04	10	07	09	05	00	15	14	02	03	12
S-caja 8															
13	02	08	04	06	15	11	01	10	09	03	14	05	00	12	07
01	15	13	08	10	03	07	04	12	05	06	11	00	14	09	02
07	11	04	01	09	12	14	02	00	06	10	13	15	03	05	08
02	01	14	07	04	10	08	13	15	12	09	00	03	05	06	11

Figura 3.11: S-cajas de DES.

En la Figura 3.13 se detalla el algoritmo que supone aplicar la P-caja. La permutación final es la inversa de IP que se encuentra detallada en la Figura 3.14.

16	07	20	21
29	12	28	17
01	15	23	26
05	18	31	10
02	08	24	14
32	27	03	09
19	13	30	06
22	11	04	25

Figura 3.12: Permutación P de la *P-caja*.

---

input: Un bloque S de 32 bits.
output: El bloque de 32 bits tras aplicar P a S.
procedure PC(S; T)
begin
01.) **for** i = 1 **to** 32 **do**
02.) T[P(i)] ← S[i]
od
return T
end

Figura 3.13: Procedimiento PC que hace actuar la P-caja sobre un bloque inicial de 32 bits

40	08	48	16	56	24	64	32	39	07	47	15	55	23	63	31
38	06	46	14	54	22	62	30	37	05	45	13	53	21	61	29
36	04	44	12	52	20	60	28	35	03	43	11	51	19	59	27
34	02	42	10	50	18	58	26	33	01	41	09	49	17	57	25

Figura 3.14: Permutación IP^{-1} de *DES*.

Desencriptado de DES

En este apartado pretendemos demostrar que un texto encriptado con DES puede ser desencriptado con el mismo algoritmo.

Definición 3.3.1. Sea $h\colon \mathbb{Z}_2^n \longrightarrow \mathbb{Z}_2^n$ una aplicación. Definimos la aplicación

$$\Phi_h\colon \mathbb{Z}_2^{2n} \longrightarrow \mathbb{Z}_2^{2n}$$

como sigue:

$$\Phi_h(x_1,\ldots,x_n,x_{n+1},\ldots,x_{2n}) = \langle x_1,\ldots,x_n \oplus h(x_{n+1},\ldots,x_{2n}),x_{n+1},\ldots,x_{2n}\rangle \tag{3.1}$$

Teorema 3.3.1. *Sea* $h\colon \mathbb{Z}_2^n \longrightarrow \mathbb{Z}_2^n$ *una aplicación. Entonces* $\Phi_h^2 = I_{\mathbb{Z}_2^{2n}}$.

Demostración. Sea $\langle x_1,\ldots,x_n,x_{n+1},\ldots,x_{2n}\rangle \in \mathbb{Z}_2^{2n}$ y llamemos $x_l = \langle x_1,\ldots,x_n\rangle$ y $x_r = \langle x_{n+1},\ldots,x_{2n}\rangle$. Se tiene que:

$$\begin{aligned}
\Phi_h^2(x_l,x_r) =& \Phi_h^2\big(x_l \oplus h(x_r),x_r\big)\\
=& \langle \big(x_l \oplus h(x_r)\big) \oplus h_{x_r},x_r\rangle\\
=& \langle x_l \oplus \big(h(x_r) \oplus h_{x_r}\big),x_r\rangle\\
=& \langle x_l \oplus \mathbf{0},x_r\rangle\\
=& \langle x_l,x_r\rangle
\end{aligned} \tag{3.2}$$

donde $\mathbf{0}$ en (3.2) representa a la $2n$-upla cuyas componentes son todas 0. □

Definición 3.3.2. Sea $x \in \mathbb{Z}_2^{32}$, $1 \leq i \leq 16$ y $K_i \in \mathbb{Z}_2^{48}$. Si E, S_i y P representan respectivamente las operaciones de la E-caja, la S-caja y la P-caja entonces definimos la aplicación:

$$T_i\colon \mathbb{Z}_2^{32} \longrightarrow \mathbb{Z}_2^{32}$$

mediante:

$$T_i(x) = P\big(S(E(x) \oplus K_i)\big) \tag{3.3}$$

Definimos además $\rho\colon \mathbb{Z}_2^{64} \longrightarrow \mathbb{Z}_2^{64}$ mediante

$$\rho(x_1,\ldots,x_{32},x_{33},\ldots,x_{64}) = \langle x_{33},\ldots,x_{64},x_1,\ldots,x_{32}\rangle \tag{3.4}$$

Definición 3.3.3. Sea $K \in \mathbb{Z}_{64}$ (la llave). Definimos la aplicación

$$DES_K : \mathbb{Z}_2^{64} \longrightarrow \mathbb{Z}_2^{64}$$

por la igualdad

$$DES_K = IP^{-1} \circ \Phi_{16} \circ \rho \circ \Phi_{15} \circ \cdots \circ \Phi_2 \circ \rho \circ \Phi_1 \circ IP \tag{3.5}$$

donde Φ_j es abreviatura de Φ_{T_j}.

Teorema 3.3.2. *Para toda* $K \in \mathbb{Z}_2^{64}$ *se cumple que:*

$$DES_K^{-1} = IP^{-1} \circ \Phi_1 \circ \rho \circ \Phi_2 \circ \cdots \circ \Phi_{15} \circ \rho \circ \Phi_{16} \circ IP \tag{3.6}$$

Como consecuencia del Teorema 3.3.2 concluimos que el algoritmo de desencriptado para *DES* es el mismo que para encriptar. La única observancia puesta de manifiesto por el teorema es la de que orden de las subllaves debe ser invertido.

Observación 3.3.1. Se ha demostrado en el año 1992 que existen llaves K_1 y K_2 para las que no existe K_3 tal que

$$DES_{K_2} \circ DES_{K_1} = DES_{K_3}$$

Esto hace más difícil el criptoanálisis de DES.

Llaves Débiles

Por el hecho de que la llave se manipula para obtener las subllaves de ronda, se puede hablar de *llaves débiles*. Una llave es débil cuando el cifrado con ella hace vulnerable al texto cifrado. Es el caso de las llaves al ser dividas en paquetes hacen que cada uno de ellos sean de ceros o de unos. Es necesario descartar las llaves débiles al encriptar, por ello es necesario su conocimiento. En muchos casos de criptosistemas están tabuladas. En [17] hay algunas referencias al criptoanálisis de *DES* que fundamentalmente se llevaría a cabo con los métodos llamados *Criptoanálisis Diferencial* (completamente explicado en [3]), *Criptoanálisis con Llaves Relacionadas* y *Criptoanálisis Lineal*. El criptoanálisis más conocido es el diferencial aunque el lineal, recientemente divulgado, se muestra algo más eficiente en algunos casos. El criptoanálisis diferencial se basa en la idea de que el criptosistema hace que se transmitan de los textos llanos a los cifrados ciertas diferencias —las diferencias aludidas son una especie de invariantes del criptosistema—. Tal propiedad permite asignar probabilidades a las llaves y como consecuencia descartar a un buen número de ellas. De todas formas el número de textos llanos que habría que estudiar en el criptoanálisis es tan elevado (del orden de 2^{47}) que *DES* se considera aún seguro.

3.4. El Sistema IDEA.

IDEA es el acrónimo de *International Data Encryption Algorithm*. Este criptosistema fue dado a conocer en 1990 por sus autores *Xuejia Lai* y *James Massey* . El algoritmo empezó llamandose *PES* —*Proposed Encryption Standard* y cuando al año siguiente

los prestigiosos maestros del criptoanálisis *Biham* y *Shamir* mostraron su método de criptoanálisis —el criptoanálisis diferencial—, los creadores de *PES* lo reforzaron frente al mismo. El resultado fue el *IPES* —*Improved Proposed Encrytion Standard* cuyo nombre fue de nuevo cambiado en 1992 adoptando el actual de *IDEA*. A juicio de muchos entendidos *IDEA* es el algoritmo criptográfico más interesante de los que se conocen.

Operaciones e Implementación.

IDEA es un cifrador de bloques que opera sobre bloques de 64 bits de texto llano. La llave es de 128 bits, se usa el mismo algoritmo para cifrar y descifrar y se sigue la filosofía de la confusión y la difusión. En el proceso no se utilizan tablas de permutación, sino que se mezcla la aritmética de varias estructuras algebraicas, siendo grupos dos de ellas:

- $\oplus$, *XOR*
- $\boxplus$, suma módulo 2^{16} (suma ignorando cualquier overflow)
- $\odot$, multiplicación módulo $2^{16}+1$ (multiplicación ignorando cualquier overflow)

todas estas operaciones actuan sobre sub-bloques de 16 bits y resultan fácilmente implementables, tanto en software como en hardware.

Definición 3.4.1. Sea $m \in \mathbb{N}$ tal que 2^m+1 es primo[2]. Consideremos la aplicación:

$$\lambda\colon \mathbb{Z}_{2^m} \longrightarrow \mathbb{Z}^*_{2^m+1}$$

definida por

$$\lambda(a) = \begin{cases} x, & \text{si } x \neq 0 \\ 2^m, & \text{si } x = 0 \end{cases} \tag{3.7}$$

Sirviéndonos de esta biyección y del producto usual de $\mathbb{Z}_{2^m+1}$ (es decir, el producto módulo 2^m+1) se define el producto $\odot$ en $\mathbb{Z}^{2^m}$ de la siguiente forma:

$$a \odot b = \lambda^{-1}\big(\lambda(a)\lambda(b) \text{ mód } (2^m+1)\big) \tag{3.8}$$

En lo sucesivo notaremos a la operación suma módulo 2^m en $\mathbb{Z}_{2^m}$ como $\boxplus$ y al opuesto de $a \in \mathbb{Z}_{2^m}$ según $\boxplus$ como $\boxminus a$. La operación XOR seguirá siendo notada por $\oplus$.

[2]Los primos de la forma 2^m+1 son denominados *primos de Fermat* y se demuestra que si 2^m+1 es primo entonces existe $k \in \mathbb{N}$tal que $m = 2^k$. Se demuestra también que no todos los números de la forma $2^{2^k}+1$ son primos.

Ejemplo 3.4.1. Para comprender las anteriores operaciones consideremos los siguientes ejemplos. $\langle \mathbb{Z}_4, \boxplus, \boxminus, 0 \rangle$ es un grupo —aditivo—, siendo la tabla de $\boxplus$ la siguiente:

$\boxplus$	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

(3.9)

Sea trata de la estructura de grupo aditivo usual en $\mathbb{Z}_4$. Si los elementos están expresados en base dos, la operación $\boxplus$ consiste en efectuar una suma con acarreo ignorando el último.

En $\mathbb{Z}_4$ la operación XOR tiene la siguiente tabla:

$\oplus$	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

(3.10)

$\langle \mathbb{Z}_5^*, \cdot, ^{-1}, 1 \rangle$ es un grupo, el grupo multiplicativo de la única estructura de cuerpo que hay sobre $\mathbb{Z}_5$. Sea la biyección λ de la Definición 3.4.1 que a los elementos 1, 2 y 3 de $\mathbb{Z}_4$ los transforma en los mismos de $\mathbb{Z}_5$ y lleva el 0 en el 4. Aprovechando el producto de $\mathbb{Z}_5$ —que notamos por la yuxtaposición— definimos la nueva operación $\odot$ en $\mathbb{Z}_4$. Su tabla es:

$\odot$	0	1	2	3
0	1	0	3	2
1	0	1	2	3
2	3	2	0	1
3	2	3	1	0

(3.11)

El álgebra $\langle \mathbb{Z}_4, \odot, ^{-1}, 1 \rangle$ es un grupo, siendo a^{-1} el inverso de a en $\mathbb{Z}_5$ para el producto $\cdot$ usual (producto módulo 5) y $0^{-1} = 0$.

Observación 3.4.1. Es sabido que $2^{16} + 1 = 65537$ es un primo de Fermat, lo cual fue aprovechado para diseñar e implementar *IDEA*. En lo que sigue nos referiremos al caso general en el que $2^m + 1$ es primo y todo lo dicho será aprovechado para el caso en que $m = 2^4 = 16$.

Teorema 3.4.1. *Sea* m *un natural tal que* $2^m + 1$ *es primo. Para todo* $a, b \in \mathbb{Z}_{2^m}$ *se cumple:*

1. *Si* $a = b = 0$ *entonces* $a \odot b = 1$.

2. *Si* $a = 0$ *y* $b \neq 0$ *entonces* $a \odot b = 2^m + 1 - b$.

3. *Si* $a \neq 0$ *y* $b \neq 0$ *entonces*

$$a \odot b = \begin{cases} (ab \bmod 2^m) - (ab \ \mathit{div}\ 2^m), & \mathit{si}\ ab \bmod 2^m \geq ab \ \mathit{div}\ 2^m, \\ (ab \bmod 2^m) - (ab \ \mathit{div}\ 2^m) + 2^m + 1, & \mathit{en\ otro\ caso}. \end{cases} \tag{3.12}$$

Demostración. Según la definición de $a \odot b$, dada por (3.8),

$$0 \odot 0 = \lambda^{-1}\left(2^{2m} \bmod (2^m + 1)\right)$$

pero, $2^{2m} = (2^m + 1)(2^m - 1) + 1$. Por tanto,

$$\begin{aligned} 0 \odot 0 &= \lambda^{-1}(1) \\ &= 1 \end{aligned}$$

Si se da el caso de que $a = 0$ y $b \neq 0$, o sea, $0 < b < 2^m$, entonces se tiene:

$$\begin{aligned} b2^m &= (2^m + 1)b - b \\ &= (2^m + 1)(b - 1) + 2^m + 1 - b \end{aligned} \tag{3.13}$$

con $0 < 2^m + 1 - b < 2^m + 1$, de donde se deduce que

$$\begin{aligned} 0 \odot b &= \lambda^{-1}(2^m + 1 - b) \\ &= 2^m + 1 - b \end{aligned}$$

Por último, supongamos que $a, b \in \mathbb{Z}_{2^m}^*$. Si

$$ab = (2^m + 1)q + r \tag{3.14}$$

con $0 \leq r \leq 2^m$, entonces se tendría también $ab = 2^m q + q + r$. Si $q + r < 2^m$, entonces

$$ab \ \mathrm{div}\ 2^m = q \tag{3.15}$$

y

$$ab \bmod 2^m = q + r \tag{3.16}$$

Si por contra $q + r \geq 2^m$, entonces se tiene:

$$\begin{aligned} ab &= 2^m q + 2^m - 2^m + q + r \\ &= 2^m(q + 1) + q + r - 2^m \end{aligned} \tag{3.17}$$

Al ser $ab \leq 2^{2m} - 1$, se tiene de (3.14) que $(2^m + 1)q + r \leq 2^{2m} - 1$. Como $r \leq 2^m$ debe darse que $(2^m + 1)q \leq 2^{2m} - 1 - r$. Así pues el máximo valor de q, q_0, se obtendrá cuando

r tenga el mínimo valor, que es 0, y dicho máximo valor cumplirá $(2^m+1)q_0 \leq 2^{2m}-1$, de donde

$$\begin{aligned}(2^m+1)q &\leq 2^{2m}-1\\ &= (2^m-1)(2^m+1)\end{aligned}$$

o sea, q debe cumplir

$$q \leq 2^m - 1 \tag{3.18}$$

De (3.18) y de (3.14) se deduce que $q+r < 2^{m+1}$. En definitiva se tiene que $q+r-2^m < 2^m$ y por tanto podemos afirmar que

$$ab \text{ div } 2^m = q+1 \tag{3.19}$$

y

$$ab \text{ mód } 2^m = q+r-2^m \tag{3.20}$$

De estas cuatro últimas igualdades podemos obtener el valor de r en función de 2^m y ab. En efecto, por (3.15) y (3.19) se cumple que $r = ab \text{ mód } 2^m - ab \text{ div } 2^m$, en el caso de que $q+r < 2^m$ y $r = ab \text{ mód } 2^m - ab \text{ div } 2^m + 2^m + 1$ en caso constrario. Es inmediato demostrar que la condición $q+r < 2^m$ es equivalente a la condición $ab \text{ mód } 2^m \geq ab \text{ div } 2^m$. Como 2^m+1 es un primo, a y b son no nulos y $a, b < 2^m+1$ resulta que $2^m+1 \nmid ab$, de donde $r \neq 0$. De ello se desprende que $\lambda^{-1}(r) = r$, o sea, que se cumple (3.12). □

Observación 3.4.2. El Teorema 3.4.1 reduce el cálculo de $a \odot b$, que por su definición estabamos obligados a efectuarlo módulo 2^m+1, a llevar a cabo las operaciones de encontrar cociente y resto de una división por 2^m. Esto último resulta muy ventajoso porque calcular resto y cociente de un número expresado en binario cuando es dividido por una potencia de dos resulta ser tomar los m dígitos menos significativos —para el resto— y desplazar hacia la derecha m posiciones los dígitos que siguen a los m menos significativos, despreciando estos últimos —para el cociente—. En definitiva, el Teorema 3.4.1 abre una vía para llevar a cabo una implementación muy eficiente de *IDEA*.

Queda por último la cuestión de cómo calcular el inverso de un elemento de Z_{2^m} para la operación $\odot$.

Teorema 3.4.2. *Sea* $m \in \mathbb{N}$ *tal que* 2^m+1 *es primo y sea* $a \in \mathbb{Z}_{2^m}$*, entonces:*

1. *Si* $a = 0$*,* $a^{-1} = 0$.
2. *Si* $a \neq 0$*, entonces* $a^{-1} = a^{-1} \text{ mód } (2^m+1)$.

Demostración. Por 1) del Teorema 3.4.1 sabemos que $0 \odot 0 = 0$, de donde $0^{-1} = 0$. En el caso de que $a \neq 0$, como $2^m + 1$ es primo, existe $b = a^{-1}$ mód $(2^m + 1)$ —calculable por el *Algoritmo de Euclides*. Es claro que $b \neq 0$, por lo que $\lambda^b = b$. Se tiene

$$\begin{aligned} \lambda^{-1}\big(\lambda(a)\lambda(b) \text{ mód } (2^m+1)\big) &= \lambda^{-1}\big(ab \text{ mód } (2^m+1)\big) \\ &= \lambda^{-1}(1) \\ &= 1 \end{aligned}$$

es decir, $a \odot \big(a^{-1} \text{ mód } (2^m + 1)\big) = 1$. Por tanto, si $a \neq 0$ entonces su inverso para $\odot$ es su inverso módulo $2^m + 1$. □

Observación 3.4.3. Del Teorema 3.4.2 se desprende que el cálculo del inverso para $\odot$ se reduce a aplicar el algoritmo de Euclides, algoritmo de considerable rapidez.

Corolario 3.4.3. *Para todo m tal que $2^m + 1$ es primo se cumple que $\langle \mathbb{Z}_{2^m}, \odot, ^{-1}, 1\rangle$ es un grupo.*

Teorema 3.4.4. *Para todo $n \in \mathbb{N}$, $\langle \mathbb{Z}_n, \boxplus, \boxminus, 0\rangle$ es un grupo.*

Algoritmo de Cifrado

El algoritmo de cifrado insite en la novedad respecto a *DES* de no utilizar permutaciones en su funcionamiento. Opera con una llave de 128 bits y 52 subllaves derivadas. Su esencia radica en la utilización de los grupos:

- $\langle \mathbb{Z}_{2^{16}}, \boxplus, \boxminus, 0\rangle$
- $\langle \mathbb{Z}_{2^{16}}, \odot, ^{-1}, 1\rangle$

El proceso de generación de las subllaves a partir de la llave inicialmente escogida es muy sencillo. Se toma la llave de 128 bits y se divide en 8 paquetes de 16 bits cada uno. Estos ocho paquetes constituyen las seis primeras llaves y las dos primeras de la segunda tanda. Seguidamente se rota a la izquierda la llave en 25 bits y se vuelve a dividir en 8 paquetes, que completan los 4 últimos que faltaban en la segunda tanda y aportan los cuatro primeros de la tercera. Se vuelve a rotar como antes la llave original y se continua así hasta obtener las 52 llaves. El proceso está esquematizado en la Figura 3.15.

input: Una palabra de 128 bits, la llave.
output: 18 listas Z^i y W^i, $1 \leq i \leq 9$, de palabras de 16 bits.
procedure LLAVES$(K; Z^i, W^i)$
begin
01.) $SK \leftarrow$ **empty**
02.) $Dividir\_en\_bloques(K, 8; \langle K_i : 1 \leq i \leq 8\rangle)$
03.) $SK \uparrow \langle K_i : 1 \leq i \leq 8\rangle$
"$\uparrow$ opera uniendo una lista a la cola de S"
04.) **for** $i = 1$ **to** 6 **do**
05.) $K \leftarrow Rot(25, K)$
"Rot rota a la izquierda K el numero de posiciones indicado"
06.) $Dividir\_en\_bloques\big(K, 8; \langle K_{8i+j} : 1 \leq j \leq 8\rangle\big)$
07.) $SK \uparrow \langle K_{8i+j} : 1 \leq j \leq 8\rangle$
od
08.) **for** $i = 1$ **to** 8 **do**
09.) **for** $j = 1$ **to** 6 **do**
10.) $Z^i[j] \leftarrow SK[6(i-1)+j]$
od
od
11.) **for** $j = 1$ **to** 4 **do**
12.) $Z^9[j] \leftarrow SK[48+j]$
od
13.) **for** $i = 1$ **to** 9 **do**
14.) **for** $j = 0$ **to** 1 **do**
15.) $W^i[4^j] \leftarrow \big(Z^{10-i}[4^j]\big)^{-1}$
od
16.) **for** $j = 2$ **to** 3 **do**
17.) $W^i[j] \leftarrow \boxminus Z^{10-i}[j]$
od
od
18.) **for** $i = 1$ **to** 8 **do**
19.) **for** $j = 5$ **to** 6 **do**
20.) $W^i[j] \leftarrow Z^{9-i}[j]$
od
od
21.) **return** $\langle Z^i : 1 \leq i \leq 9\rangle, \langle W^i : 1 \leq i \leq 9\rangle$
end

Figura 3.15: Procedimiento LLAVES para generar las 52 subllaves que necesita IDEA.

input: Una palabra de 64 bits P de texto llano y una llave K de 128 bits.
output: Una palabra de 64 bits C de texto cifrado.
procedure IDEA$(P, Z^1, \ldots, Z^9; C)$
    begin
01.) Dividir$(P; X_1, \ldots, X_4)$
02.) $i \leftarrow 1$
    "$Z^r[j]$ es la llave j de la vuelta r"
03.) **while** $i \leq 8$ **do**
04.) $W_1 \leftarrow Z^i[1] \odot X_1$, $W_2 \leftarrow Z^i[2] \boxplus X_2$, $W_3 \leftarrow Z^i[3] \boxplus X_3$, $W_4 \leftarrow Z^i[4] \odot X_4$
05.) $V_1 \leftarrow W_1 \oplus W_3$, $V_2 \leftarrow W_2 \oplus W_4$
06.) $Y_1 \leftarrow V_1 \odot Z^i[5]$, $Y_2 \leftarrow Y_1 \boxplus V_2$
07.) $R_1 \leftarrow Y_2 \odot Z^i[6]$, $R_2 \leftarrow Y_1 \boxplus R_1$
08.) $X_1 \leftarrow W_1 \oplus R_1$, $X_2 \leftarrow W_2 \oplus R_2$, $X_3 \leftarrow W_3 \oplus R_1$, $X_4 \leftarrow W_4 \oplus R_2$
09.) $Q \leftarrow X_3$, $X_3 \leftarrow X_2$, $X_2 \leftarrow Q$
10.) $i \leftarrow i + 1$
    od
11.) **for** $i = 1$ **to** 4 **do**
12.) **if** $i \in \{2, 3\}$ **then** $C_i \leftarrow X_i \boxplus Z^9[i]$
13.) **else** $C_i \leftarrow X_i \odot Z^9[i]$
    od
14.) $C \leftarrow$ Unir$(C_1, \ldots, C_4)$
15.) **return** C
    end

Figura 3.16: Procedimiento IDEA para cifrar un bloque de texto llano de 64 bits según el algoritmo "*International Data Encryption Standard*".

Algoritmo de Descifrado

Para descifrar un texto encriptado por *IDEA* es preciso el suministro de las llaves, sus opuestas o inversas en un orden adecuado que viene a ser, en cierto sentido, el inverso del que se usó para cifrar.

Definición 3.4.2. Sean $a, b \in \mathbb{Z}_{2^{16}}$. Definimos las funciones:

$$\delta^i_{\langle a,b\rangle} \colon \mathbb{Z}_{2^{16}} \times \mathbb{Z}_{2^{16}} \longrightarrow \mathbb{Z}_{2^{16}}$$

para $i = 1, 2$ por las siguientes igualdades:

$$\begin{aligned} \delta^1_{\langle a,b\rangle}(u, v) &= \big((u \odot a) \boxplus v\big) \odot b \\ \delta^2_{\langle a,b\rangle}(u, v) &= \big(\big((u \odot a) \boxplus v\big) \odot b\big) \boxplus (u \odot a) \end{aligned} \tag{3.21}$$

Finalmente definimos la función:

$$\delta_{\langle a,b\rangle} \colon \mathbb{Z}_{2^{16}} \times \mathbb{Z}_{2^{16}} \longrightarrow \mathbb{Z}_{2^{16}} \times \mathbb{Z}_{2^{16}}$$

por la igualdad:

$$\delta_{\langle a,b\rangle}(u, v) = \langle \delta^1_{\langle a,b\rangle}(u, v), \delta^2_{\langle a,b\rangle}(u, v)\rangle \tag{3.22}$$

Definición 3.4.3. Sean $a, b \in \mathbb{Z}_{2^{16}}$ las funciones:

$$\varphi^i_{\langle a,b\rangle} \colon \mathbb{Z}_2^{16} \times \overset{4)}{\cdots} \times \mathbb{Z}_2^{16} \longrightarrow \mathbb{Z}_2^{16}$$

para $1 \leq i \leq 4$ definidas para todo $\langle x_1, \ldots, x_4\rangle \in \mathbb{Z}_2^{16} \times \overset{4)}{\cdots} \times \mathbb{Z}_2^{16}$ como:

1. $\varphi^1_{\langle a,b\rangle}(x_1, \ldots, x_4) = \delta^1_{\langle a,b\rangle}(x_1 \oplus x_3, x_2 \oplus x_4) \oplus x_1$
2. $\varphi^2_{\langle a,b\rangle}(x_1, \ldots, x_4) = \delta^2_{\langle a,b\rangle}(x_1 \oplus x_3, x_2 \oplus x_4) \oplus x_2$
3. $\varphi^3_{\langle a,b\rangle}(x_1, \ldots, x_4) = \delta^1_{\langle a,b\rangle}(x_1 \oplus x_3, x_2 \oplus x_4) \oplus x_3$
4. $\varphi^4_{\langle a,b\rangle}(x_1, \ldots, x_4) = \delta^2_{\langle a,b\rangle}(x_1 \oplus x_3, x_2 \oplus x_4) \oplus x_4$

A partir de estas cuatro funciones definimos una nueva:

$$\varphi_{\langle a,b\rangle} \colon \mathbb{Z}_2^{16} \times \overset{4)}{\cdots} \times \mathbb{Z}_2^{16} \longrightarrow \mathbb{Z}_2^{16} \times \overset{4)}{\cdots} \times \mathbb{Z}_2^{16}$$

mediante la igualdad:

$$\begin{aligned} \varphi_{\langle a,b\rangle}(x_1, \ldots, x_4) = \langle &\varphi^1_{\langle a,b\rangle}(x_1, \ldots, x_4), \varphi^2_{\langle a,b\rangle}(x_1, \ldots, x_4), \\ &\varphi^3_{\langle a,b\rangle}(x_1, \ldots, x_4), \varphi^4_{\langle a,b\rangle}(x_1, \ldots, x_4)\rangle \end{aligned} \tag{3.23}$$

Lema 3.4.5. *Para todo* $a, b \in \mathbb{Z}_{2^{16}}$ *se cumple que* $\varphi^2_{\langle a,b\rangle} = I$.

Observación 3.4.4. En la demostración del Lema 3.4.5 aparece claro que el único papel de $\delta_{\langle a,b\rangle}$ es el de "difundir". Esta función podría ser sustituida por cualquier otra "caja difusora" dentro del esquema de *IDEA*.

Lema 3.4.6. *La función*

$$\iota\colon \mathbb{Z}_2^{16} \times \overset{4)}{\cdots} \times \mathbb{Z}_2^{16} \longrightarrow \mathbb{Z}_2^{16} \times \overset{4)}{\cdots} \times \mathbb{Z}_2^{16}$$

definida por

$$\iota(x_1, x_2, x_3, x_4) = \langle x_1, x_3, x_2, x_4\rangle \tag{3.24}$$

cumple la condición $\iota^2 = I$.

Teorema 3.4.7. *Sea* P *una palabra de 64 bits. Para toda:*

1. *Llave* K *de 128 bits*
2. *Llaves* Z^i $(1 \leq i \leq 9)$ *de cifrar generadas por el procedimiento de la Figura 3.15.*
3. *Llaves* W^i $(1 \leq i \leq 9)$ *de descifrar generadas por el procedimiento de la Figura 3.15.*

se cumple que:

$$\mathrm{IDEA}(\mathrm{IDEA}(P, Z^1, \ldots, Z^9; C_1), W^1, \ldots, W^9; C_2) = P$$

3.5. Modos de Utilizar un Cifrador por Bloques

Casi ningún sistema de cifrado por bloques se usa por sí solo. Usualmente se recurre a modos de utilizarlo más allá que el empleo estricto del algoritmo. En lo que sigue hacemos una concisa exposición de algunos modos de cifrar, entre ellos: *ECB*, *CBC*, *CFB* y *OFB*.

Libro de Códigos Electrónico (*ECB*)

La forma más obvia de utilización es como cifrador de bloques de texto llano. Un bloque concreto se cifrará de la misma forma siempre, por lo que se podría considerar que lo que

se tiene realmente para cada llave es un dicciónario electrónico o un *libro de códigos electrónico —electronic codebook* en Inglés y de ahí la denominación *ECB*—. Este hipotético diccionario tendría 2^{64} entradas si se cifraran bloques en bloques de 64 bits. Es claro que por este método cada bloque de texto llano se cifra sin más consideración que él mismo.

De utilizar este método de enciframiento, por ejemplo para ocultar información de una base de datos o para hacer seguro el correo electrónico, debemos cifrar los bloques del texto llano en un orden que no sea el natural. Ello hará difícil que en manos del atacante caiga una excesiva cantidad de texto llano y su correspondiente cifrado. Se pueden cifrar algunos bloques centrales, luego los finales y luego los del principio.

---

input: Un texto llano dividido en m bloques P_i, $1 \leq i \leq m$.
output: El texto cifrado correspondiente dividido en m bloques C_i, $1 \leq i \leq m$.
requisito: Un bloque inicial I_0 del tamaño de los bloques de texto y un criptosistema E_K.
procedure $\mathrm{CBC}\big(E_K, I_0, \langle P_i : 1 \leq i \leq m\rangle; \langle C_i : 1 \leq i \leq m\rangle\big)$
    begin
01.) $C[0] \leftarrow I_0$
02.) $i \leftarrow 1$
03.) **while** $i \leq m$ **do**
04.) $C[i] \leftarrow E_K\big(P[i] \oplus C[i-1]\big)$
05.) $i \leftarrow i + 1$
    od
06.) **return** $\langle C[i] : 1 \leq i \leq m\rangle$
    end

Figura 3.17: Modo CBC de utilizar un criptosistema.

Encadenamiento de Bloques Cifrados

Junto al método *ECB* de utilización de *DES*, *ANSI* prescribe la utilización del método *CBC* de cifrado. *CBC* son las iniciales de *Cipher Block Chaining*, que en Español vendría a ser *Encadenamiento de Bloques Cifrados*.

La forma *CBC* de utilizar un sistema de cifrado está basada en realimentar el algoritmo de cifrado con el resultado de cifrar el bloque anterior. El bloque es utilizado para modificar el cifrado del siguiente, de forma que a un bloque puede corresponder más de una cifra. El método consiste en operar mediante la operación booleana $\oplus$ el bloque de

texto llano a cifrar con el resultado de cifrar el anterior. En la Figura 3.17 se especifica el método.

Para descifrar utilizamos la misma idea que para cifrar. El método aparece explicitado en la Figura 3.18.

---

input: Un texto cifrado dividido en m bloques C_i, $1 \leq i \leq m$.
output: El texto llano correspondiente dividido en m bloques P_i, $1 \leq i \leq m$.
requisito: El bloque inicial I_0 utilizado para cifrar.
procedure $\mathrm{CBC}^{-1}\big(E_K, I_0, \langle C_i\colon 1 \leq i \leq m\rangle; \langle P_i\colon 1 \leq i \leq m\rangle\big)$
begin
01.) $C[0] \leftarrow I_0$
02.) $i \leftarrow 1$
03.) **while** $i \leq m$ **do**
04.) $\quad P[i] \leftarrow C[i-1] \oplus E_K\big(C[i]\big)$
05.) $\quad i \leftarrow i + 1$
od
06.) **return** $\langle P[i]\colon 1 \leq i \leq m\rangle$
end

Figura 3.18: Descifrar un texto encriptado con el método CBC.

Cuando se utiliza el método *CBC* de cifrado ocurre que textos llanos idénticos hasta cierta línea se encriptan de idéntica forma hasta esa línea. Es el caso de cuando se están encriptando textos con una cabecera común: cartas confidenciales, fichas de una base de datos, correos electrónicos, etc. Estas redundancias suponen información añadida a un posible e indeseable oponente.

Para atenuar este peligro procedemos encriptando en primer lugar no el primer bloque, sino un bloque generado aleatoriamente y que puede ser llamado *vector incial* (IV). El sentido de introducir este bloque —aleatorio y sin sentido— es el de hacer único cada mensaje. El usuario autorizado que desencripta debe proceder como si este bloque aleatorio fuera parte del texto, descifrar e ignorarlo en la lectura del texto llano.

El mensaje espurio IV no es necesario mantenerlo secreto, se envia encriptado con el mensaje. Sin embargo, es preciso cambiarlo con cada mensaje. En [17] pueden encontrarse diversos métodos de utilizar los criptosistemas.

Cifrado con Realimentación (*CFB*)

El *cifrado con realimentación* —en Inglés *cipher feedback* (*CFB*)— tiene como propiedad importante la de que el cifrado se realiza en unidades menores que la del tamaño de bloque, por ejemplo un byte. Pero no hay nada de especial en el número 8, si se deseara se podía cifrar un bit en cada momento utilizando un 1-bit *CFB*. Nos referiremos al caso de un 8-bit *CFB* operando sobre la base de un algoritmo de cifrado por bloques de 64 bits.

La idea consiste en disponer de una fila de 8 bytes, que estará inicializada a un valor *IV* escogido y proceder en cada paso como sigue:

1. Se toma el bloque de 64 bits que constituye la fila y se encripta.
2. Se toma el byte más a la izquierda del bloque que resulta de cifrar y se opera mediante la función $\oplus$ con el primer byte de texto llano.
3. El byte que resulta de la operación anterior es el primer byte de texto cifrado.
4. Este byte se coloca en la cola de la fila y su cabeza es desechada.

Este proceso es descrito en el algoritmo de la Figura 3.19.

Para descifrar un texto encriptado con el método *CFB* y según el criptosistema de bloques E_K, utilizamos el procedimiento inverso. Dicho procedimiento está esquematizado en la Figura 3.20.

Observación 3.5.1. En esta peculiar forma de cifrar se observa que nunca se utiliza el algoritmo de cifrar en su configuración de descifrar. Por otra parte, el modo *CFB* es muy adecuado para utilizarlo en redes seguras. Obsérvese que en la forma en que se ha descrito, podríamos realizar las operaciones:

- escribir un carácter de texto llano,
- cifrarlo y
- enviarlo.

y ello carácter a carácter. De esta forma no habría que esperar a que el texto llano estuviera escrito sino que se cifraría y se transmitiría conforme se va escribiendo. La lista de 8 bytes inciciales debe ser enviada junto a la llave.

input: Un texto llano dividido en m bloques de un byte P_i, $1 \leq i \leq m$.
output: El texto cifrado correspondiente dividido en m bloques de un byte C_i, $1 \leq i \leq m$.
requisito: Un criptosistema por bloques de 68 bits E_K y un bloque inicial I_0 dividido en paquetes de 1 byte.

procedure CFB$\big(E_K, I_0, \langle P_i : 1 \leq i \leq m\rangle; \langle C_i : 1 \leq i \leq m\rangle\big)$
    begin
01.) $L \leftarrow I_0$
02.) $i \leftarrow 1$
03.) **while** $1 \leq m$ **do**
04.) $M \leftarrow E_K(L)$
05.) $Y \leftarrow M[1]$
        "Y es un byte porque M es una lista de bytes"
06.) $C_i \leftarrow Y \oplus P_i$
07.) **for** $j = 2$ **to** 8 **do**
08.) $L[j-1] \leftarrow L[j]$
        od
09.) $L[8] \leftarrow C_i$
10.) $i \leftarrow i + 1$
    od
11.) **return** $\langle C_i : 1 \leq i \leq m\rangle$
    end

Figura 3.19: Modo CFB de utilizar un criptosistema.

input: Un texto cifrado dividido en m bloques C_i de un byte, $1 \leq i \leq m$.
output: El texto llano correspondiente dividido en m bloques P_i de un byte, $1 \leq i \leq m$.
requisito: El bloque inicial I_0 utilizado para cifrar.
procedure $\mathrm{CFB}^{-1}\big(E_K, I_0, \langle C_i : 1 \leq i \leq m\rangle; \langle P_i : 1 \leq i \leq m\rangle\big)$
begin
01.) $L \leftarrow I_0$
02.) $i \leftarrow 1$
03.) **while** $1 \leq m$ **do**
04.) $M \leftarrow E_K(L)$
05.) $Y \leftarrow M[1]$
"Y es un byte porque M es una lista de bytes"
06.) $P_i \leftarrow Y \oplus C_i$
07.) **for** $j = 2$ **to** 8 **do**
08.) $L[j-1] \leftarrow L[j]$
od
09.) $L[8] \leftarrow C_i$
10.) $i \leftarrow i + 1$
od
11.) **return** $\langle P_i : 1 \leq i \leq m\rangle$
end

Figura 3.20: Descifrar un texto encriptado con el método *CFB*.

Cifrado con Realimentación Interna (*OFB*)

Este sistema de cifrado es idéntico en esencia al CFB salvo en como se lleva a cabo la realimentación de la fila. El proceso es como sigue:

La idea ahora consiste en disponer de una fila de 8 bytes, que estará inicializada a un valor IV escogido y proceder en cada paso como sigue:

1. Se toma el bloque de 64 bits que constituye la fila y se encripta.
2. Se toma el byte más a la izquierda del bloque que resulta de cifrar y es operado mediante la función $\oplus$ con el primer byte de texto llano.
3. El byte que resulta de la operación anterior es el primer byte de texto cifrado.
4. El byte que se operó con el byte de texto llano se hace pasar a la cola de la fila y ésta se desplaza a la izquierda una posición.

Este proceso es descrito en el algoritmo de la Figura 3.21.

input: Un texto llano dividido en m bloques de un byte P_i, $1 \leq i \leq m$.
output: El texto cifrado correspondiente dividido en m bloques de un byte C_i, $1 \leq i \leq m$.
requisito: Un criptosistema por bloques de 68 bits E_K y un bloque inicial I_0 dividido en paquetes de 1 byte.

procedure OFB$\big(E_K, I_0, \langle P_i : 1 \leq i \leq m\rangle; \langle C_i : 1 \leq i \leq m\rangle\big)$
    begin
01.) $L \leftarrow I_0$
02.) $i \leftarrow 1$
03.) **while** $1 \leq m$ **do**
04.) $M \leftarrow E_K(L)$
05.) $Y \leftarrow M[1]$
        "Y es un byte porque M es una lista de bytes"
06.) $C_i \leftarrow Y \oplus P_i$
07.) **for** $j = 2$ **to** 8 **do**
08.) $L[j-1] \leftarrow L[j]$
        od
09.) $L[8] \leftarrow Y$
10.) $i \leftarrow i + 1$
    od
11.) **return** $\langle C_i : 1 \leq i \leq m\rangle$
    end

Figura 3.21: Modo OFB de utilizar un criptosistema.

input: Un texto cifrado dividido en m bloques C_i de un byte, $1 \leq i \leq m$.
output: El texto llano correspondiente dividido en m bloques P_i de un byte, $1 \leq i \leq m$.
requisito: El bloque inicial I_0 utilizado para cifrar.
procedure $\text{OFB}^{-1}(E_K, I_0, \langle C_i : 1 \leq i \leq m\rangle; \langle P_i : 1 \leq i \leq m\rangle)$
begin
01.) $L \leftarrow I_0$
02.) $i \leftarrow 1$
03.) **while** $1 \leq m$ **do**
04.) $M \leftarrow E_K(L)$
05.) $Y \leftarrow M[1]$
"Y es un byte porque M es una lista de bytes"
06.) $P_i \leftarrow Y \oplus C_i$
07.) **for** $j = 2$ **to** 8 **do**
08.) $L[j-1] \leftarrow L[j]$
od
09.) $L[8] \leftarrow Y$
10.) $i \leftarrow i + 1$
od
11.) **return** $\langle P_i : 1 \leq i \leq m\rangle$
end

Figura 3.22: Descifrar un texto encriptado con el método *OFB*.

Capítulo 4

El Criptosistema AES

La descripción técnica de AES que aportamos aquí está fundada en el documento del NIST (National Institute of Standards and Technology) con el nombre NIST.FIPS.197.pdf.[1]

4.1. Glosario

Para el estándar AES es considerada la siguiente terminología:

- AES; acrónimo de "Advanced Encryption Standard".
- Array; lista ordenada y numerada de entidades idénticas, que pueden ser bits o bytes.
- Bit; dígito binario representado como 0 ó 1.
- Bloque; lista de bits que representan la entrada, salida, estado y clave de ronda. La longitud del bloque es el número de bits de que se compone. Puede ser entendido también como un array de bytes.
- Byte; agrupación de ocho bits, tratado como una entidad aislada o como un array de 8 bits.
- Cifrador; serie de transformaciones que convierten el texto llano en texto cifrado utilizando la clave de cifrado.
- Cifrado inverso; serie de transformaciones que convierten texto cifrado en plano usando la clave de cifrado.

[1] FIPS es abreviatura de Federal Information Processing Standards.

- Clave de cifrado; clave criptográfica secreta que es usada por el mecanismo (rutina) de expansión de claves para generar un conjunto de claves de ronda. Puede ser representada como un array rectangular de bytes, con cuatro filas y Nk columnas.
- Clave de ronda; las claves de ronda son valores derivados de la clave de cifrado utilizando la rutina de expasión de claves; son aplicadas al estado en el cifrador y el cifrador inverso.
- Estado; resultado intermedio del cifrador que puede ser representado por un array de bytes rectangular, tiene cuatro filas y Nb columnas.
- Expansión de la clave; rutina usada para generar una serie de claves de ronda a partid de la clave de cifrado.
- Nb; número de columnas (palabras de 32 bits (32 bits words)) que componen el Estado. Para este estándar Nb=4.
- Nk; número de palabras de 32 bits (32 bits words) que forman la clave de cifrado. Para este estándar, Nk $= 4$, 6 ó 8.
- Nr; Número de rondas, que es una función de Nk y Nb (los cuales están fijos). Para este estándard, Nr $= 10$, 12 ó 14.
- Palabra (word); un grupo de 32 bits que es tratado como una entidad o un array de 4 bytes.
- Transformación afín; una transformación consistente en la multiplicación por una matriz seguida de la suma de un vector.
- Texto cifrado; salida del cifrador o entrada del cifrador inverso.
- Texto llano; entrada del cifrador o salida del cifrador inverso.
- Rijndael; algoritmo criptográfico especificado en este Advanced Encryption Standard (AES).
- S-caja; tabla de sustitución no lineal utilizada en varias transformaciones de sustitución de bytes y en la rutina de expansión para llevar a cabo una sustitución uno-a-uno de un valor de byte.
- `xor`; operación de la o excluiva.
- $\oplus$; representación de la `xor`.
- $\otimes$; multiplicación de dos polinomios, cada uno de ellos de grado inferior a 4, módulo $x^4 + 1$.
- $\bullet$; multiplicación en cuerpos finitos.

4.2. Estructuras Básicas

La unidad básica de proceso en AES es también el bit, el byte y el word (palabra):

- bit; dígito binario con valor 0 ó 1.
- byte; grupo de ocho bits que es tratado como una entidad aislada o como una secuencia de 8 bits individuales $b_7b_6b_5b_4b_3b_2b_1b_0$. El byte puede ser expresado como:
  - Una lista de 8 bits simplemente, p.e., 01101110.
  - Dos dígitos de la codificación hexadecimal (cfr. Fig. 4.1), p.e., 6e es el equivalente a 01101110, entendido como 0110 1110.

patrón de bits	carácter	patrón de bits	carácter
0000	0	1000	8
0001	1	1001	9
0010	2	1010	a
0011	3	1011	b
0100	4	1100	c
0101	5	1101	d
0110	6	1110	e
0111	7	1111	f

Figura 4.1: Representación hexadecimal de los patrones de bits.

  - Codificando al polinomio $\sum_{i=0}^{7} b_i x^i$, p.e., 01101110 corresponde a $x^6 + x^5 + x^3 + x^2 + x$.
- Palabra (word); un grupo de 32 bits, tratado como una entidad aislada o como una secuencia de 4 bytes.
- Array de bytes; representado en la forma $a_0a_1 \ldots a_{15}$ y considerado eventualmente como una lista de bits, cada uno con su correspondiente subíndice.
- Estado (State); es un array bidimensional de bytes con cuatro columnas cada una conteniendo Nb bytes, donde Nb es la longitud del bloque dividida por 32. En este estándar el bloque es de 128 bits, por lo que Nb=4. El paquete de 16 bytes entrante, $in_0, in_1, \ldots, in_{15}$ es organizado en un array bidimensional para su transformación y pasar a generar el paquete saliente $out_0, out_1, \ldots, out_{15}$. Puede

entenderse como sigue:

bytes entrantes

in_0	in_4	in_8	in_{12}
in_1	in_5	in_9	in_{13}
in_2	in_6	in_{10}	in_{14}
in_3	in_7	in_{11}	in_{15}

$\rightarrow$ estado

s_{00}	s_{01}	s_{02}	s_{03}
s_{10}	s_{11}	s_{12}	s_{13}
s_{20}	s_{21}	s_{22}	s_{23}
s_{30}	s_{31}	s_{32}	s_{33}

$\rightarrow$ bytes salientes

out_0	out_4	out_8	out_{12}
out_1	out_5	out_9	out_{13}
out_2	out_6	out_{10}	out_{14}
out_3	out_7	out_{11}	out_{15}

Así pues, para todo $0 \leq r < 4$ y $0 \leq c < Nb$,

$$in_{r+4c} = s_{rc} = out_{r+4c}$$

Los cuatro bytes en cada columna de un estado (state) forman una palabra (word) de 32-bits, donde el número de la columna proporciona el índice a la palabra:

$$w_0 = s_{00}s_{10}s_{20}s_{30} \qquad w_2 = s_{02}s_{12}s_{22}s_{32}$$
$$w_1 = s_{01}s_{11}s_{21}s_{31} \qquad w_3 = s_{03}s_{13}s_{23}s_{33}$$

Por tanto, el estado puede ser interpretado también como un array unidimensional de palabras de 32 bits (columnas), a saber: w_0, w_1, w_2 y w_3, donde el índice de la columna provee el índice en las entradas del array.

4.3. AES y los Cuerpos Finitos

Buena parte del sustento de AES viene provisto por la aritmética en determinado cuerpo finito. En esta sección presentamos y analizamos ese cuerpo. También podremos ejemplos que resultarán relevantes posteriormente.

Lema 4.3.1. *Para todo número natural* n*, en* $\mathbb{Z}_2[x]$ *hay* 2^n *polinomios no nulos de grado* n*.*

Lema 4.3.2. *Sea* A *un dominio de integridad y* $p(x)$ *un polinomio no nulo de* $A[x]$*. Si* $p(x)$ *es reducible entonces tiene un factor de grado menor o igual que* d*, donde:*

$$d = \begin{cases} m/2, & \text{si } m \equiv 0 \pmod 2 \\ (m-1)/2, & \text{si } m \equiv 1 \pmod 2 \end{cases}$$

Lema 4.3.3. *En* $\mathbb{Z}_2[x]$ *el único polinomio irreducible de grado* 2 *es* $x^2 + x + 1$*.*

Demostración. Los 4 polinomios de grado 2 de $\mathbb{Z}_2[x]$ son:

- $p_0(x) = x^2 = xx$

- $p_1(x) = x^2 + 1 = (x+1)^2$
- $p_2(x) = x^2 + x = x(x+1)$ y
- $p_3(x) = x^2 + x + 1$, para el que $p_3(0) = 1 = p_1(x)$.

Por tanto, $p_3(x)$ es irreducible y para todo $0 \leq k \leq 2$, $p_k(x)$ es reducible. □

Lema 4.3.4. *En* $\mathbb{Z}_2[x]$ *los únicos polinomios irreducibles de grado* 3 *son* $x^3 + x^2 + 1$ *y* $x^3 + x + 1$.

Demostración. Los 8 polinomios de grado 3 de $\mathbb{Z}_2[x]$ son:

- $p_0(x) = x^3$
- $p_1(x) = x^3 + 1 = (x+1)(x^2 + x + 1)$
- $p_2(x) = x^3 + x = x(x+1)^2$
- $p_3(x) = x^3 + x + 1$, para el que $p_3(0) = 1 = p_1(x)$.
- $p_4(x) = x^3 + x^2 = x^2(x+1)$,
- $p_5(x) = x^3 + x^2 + 1$, para el que $p_5(0) = 1 = p_1(x)$.
- $p_6(x) = x^3 + x^2 + x = x(x^2 + x + 1)$,
- $p_7(x) = x^3 + x^2 + x + 1 = (x+1)^3$,

Por tanto, $p_3(x)$ y $p_5(x)$ son irreducible y para todo $k \in 8 \setminus \{3, 5\}$, $p_k(x)$ es reducible. □

Lema 4.3.5. *En* $\mathbb{Z}_2[x]$ *los únicos polinomios irreducibles de grado* 4 *son* $x^4 + x^3 + x^2 + x + 1$, $x^4 + x^3 + 1$ *y* $x^4 + x + 1$.

Demostración. Los 16 polinomios de grado 4 de $\mathbb{Z}_2[x]$ son:

- $p_0(x) = x^4$,
- $p_1(x) = x^4 + 1 = (x+1)^4$,
- $p_2(x) = x^4 + x = x(x+1)(x^2 + x + 1)$,
- $p_3(x) = x^4 + x + 1$, cumple $p_3(0) = 1 = p_1(x)$ y no tener como factor a $x^2 + x + 1$,
- $p_4(x) = x^4 + x^2 = x^2(x+1)^2$,

- $p_5(x) = x^4 + x^2 + 1 = (x^2 + x + 1)^2$,
- $p_6(x) = x^4 + x^2 + x = x(x^3 + x + 1)$,
- $p_7(x) = x^4 + x^2 + x + 1 = (x + 1)(x^3 + x^2 + 1)$,
- $p_8(x) = x^4 + x^3 = (x + 1)x^3$,
- $p_9(x) = x^4 + x^3 + 1$, cumple $p_9(0) = 1 = p_9(x)$ y no tener como factor a $x^2 + x + 1$,
- $p_{10}(x) = x^4 + x^3 + x = x(x^3 + x^2 + 1)$,
- $p_{11}(x) = x^4 + x^3 + x + 1 = (x + 1)^2(x^2 + x + 1)$,
- $p_{12}(x) = x^4 + x^3 + x^2 = x^2(x^2 + x + 1)$,
- $p_{13}(x) = x^4 + x^3 + x^2 + 1 = (x + 1)(x^3 + x + 1)$,
- $p_{14}(x) = x^4 + x^3 + x^2 + x = x(x + 1)^3$ y
- $p_{15}(x) = x^4 + x^3 + x^2 + x + 1$, cumple $p_{15}(0) = 1 = p_{15}(x)$ y no tener como factor a $x^2 + x + 1$.

Por tanto, $p_3(x)$, $p_9(x)$ y $p_{15}(x)$ son irreducible y para todo $k \in 16 \setminus \{3, 9, 15\}$, $p_k(x)$ es reducible. □

Resumiendo y añadiendo los polinomios de $\mathbb{Z}_2[x]$ irreducibles con grado igual a 5:

- x
- $x + 1$
- $x^2 + x + 1$
- $x^3 + x + 1$
- $x^3 + x^2 + 1$
- $x^4 + x + 1$
- $x^4 + x^3 + 1$
- $x^4 + x^3 + x^2 + x + 1$
- $x^5 + x^2 + 1$
- $x^5 + x^3 + 1$

- $x^5 + x^3 + x^2 + x + 1$
- $x^5 + x^4 + x^2 + x + 1$
- $x^5 + x^4 + x^3 + x + 1$
- $x^5 + x^4 + x^3 + x^2 + 1$

Definición 4.3.1. Nombraremos al polinomio $x^8 + x^4 + x^3 + x + 1$ de $\mathbb{Z}_2[x]$ por $m(x)$.

Teorema 4.3.6. *El polinomio* $m(x)$*, es decir:*

$$x^8 + x^4 + x^3 + x + 1$$

es irreducible como polinomio de $\mathbb{Z}_2[x]$*.*

Demostración. Acontece que:

- $m(0) = 1 = m(1)$,
- $m(x) = (x^2 + x + 1)(x^6 + x^5 + x^3) + (x + 1)$,
- $m(x) = (x^3 + x + 1)(x^5 + x^3 + x^2 + 1) + x^2$,
- $m(x) = (x^3 + x^2 + 1)(x^5 + x^4 + x^3) + (x + 1)$,
- $m(x) = (x^4 + x + 1)(x^4 + x) + (x^3 + x^2 + 1)$,
- $m(x) = (x^4 + x^3 + 1)(x^4 + x^3 + x^2 + x + 1) + (x^3 + x^2)$ y
- $m(x) = (x^4 + x^3 + x^2 + x + 1)(x^4 + x^3 + 1) + (x^3 + x^2)$.

de donde $m(x)$ no tiene ningún factor de grado menor o igual que 4 y, como consecuencia, no puede ser reducible; es por tanto irreducible □

Corolario 4.3.7. *Para todo polinomio* $n(x) \in \mathbb{Z}_2[x]^*$ *tal que* $\deg(n(x)) < 8$*, existe y es posible encontrar* $u(x)$ *tal que:*

1. $\deg(u(x)) < 8$
2. $u(x)n(x) \equiv 1 \pmod{m(x)}$

Demostración. Al ser $\mathbb{Z}_2$ un cuerpo, $\mathbb{Z}_2[x]$ es un dominio euclídeo. El algoritmo extendido de Euclides proporciona polinomios $u(x)$ y $v(x)$ tales que:

$$u(x)n(x) + v(x)m(x) = g(x) \tag{4.1}$$

donde $g(x) = (n(x), m(x))$ (máximo común divisor de $n(x)$ y $m(x)$). Según establece el Teorema 4.3.6, $m(x)$ es irreducible y por tanto $g(x) = 1$. Así pues, de (4.1) se deduce que:

$$u(x)n(x) \equiv 1 \pmod{m(x)}$$

y esta misma relación la cumple $u(x)$ mód $m(x)$ por lo que podemos considerar que $\deg(u(x)) < \deg(m(x)) = 8$. □

Observación 4.3.1. Por sus propiedades artiméticas, el polinomio $u(x)$ de la demostración del Corolario 4.3.7 —reducido módulo $m(x)$ si fuese necesario— es representado como $n(x)^{-1}$ mód $m(x)$ y denominado el *inverso* de $n(x)$ módulo $m(x)$.

Corolario 4.3.8. *El anillo de polinomios:*

$$\mathbb{Z}_2[x]/(x^8 + x^4 + x^3 + x + 1)$$

es isomorfo al cuerpo $GF(2^8)$.

Observación 4.3.2. Los elementos de $GF(2^8)$ serán manipulados en adelante representándolos de algunas de las siguientes formas:

- Por el único representante de su clase que tiene grado inferior a 8.
- Por la representación binaria que corresponde a la lista de los coeficientes del polinomio antes nombrado.
- Por la representación hexadecimal de la representación binaria antes nombrada (cfr. Figura 4.1)

Por ejemplo, el elemento $[x^{13}+x^{11}+x^9+x^5+x^4+x^3+1]$ será representado por x^7+x^6+1 y consencuentemente por 11000001 y c1 (c es por 1100 y 1 es por 0001); la razón es que:

$$x^{13} + x^{11} + x^9x^8 + x^6 + x^5 + x^4 + x^3 + 1 \equiv x^7 + x^6 + 1 \pmod{m(x)}$$

ya que por el algoritmo de la división:

$$x^{13} + x^{11} + x^9x^8 + x^6 + x^5 + x^4 + x^3 + 1 = (x^5 + x^3)m(x) + (x^7 + x^6 + 1)$$

Ejemplo 4.3.1. En $GF(2^8)$ representado como $\mathbb{Z}_2[x]/(x^8+x^4+x^3+x+1)$ efectuamos seguidamente la suma de $x^6+x^4+x^2+x+1$ y x^7+x+1 según las tres modalidades notacionales:

$$\begin{aligned}(x^6+x^4+x^2+x+1)+(x^7+x+1) &= x^7+x^6+x^4+x^2\\ \{01010111\}\oplus\{10000011\} &= \{11010100\}\\ \{57\}\oplus\{83\} &= \{d4\}\end{aligned}$$

e igualmente el producto

$$\begin{aligned}(x^6+x^4+x^2+x+1)(x^7+x+1) &= x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1\\ &(\equiv (x^7+x^6+1) \pmod{m(x)})\\ &= x^7+x^6+1\end{aligned}$$

por lo que:

$$\begin{aligned}(x^6+x^4+x^2+x+1)(x^7+x+1) &= x^7+x^6+1\\ \{01010111\}\bullet\{10000011\} &= \{11000001\}\\ \{57\}\bullet\{83\} &= \{c1\}\end{aligned}$$

Ejemplo 4.3.2. Consideremos $\{53\}$ como representación de $n(x)$, esto es, $n(x) = x^6 + x^4 + x + 1$. Se tiene que:

$$(x^7+x^6+x^3+x)n(x)+(x^5+x^4+x^3+x^2+1)m(x) = 1$$

luego $n(x)^{-1} \pmod{m(x)} = x^7+x^6+x^3+x$. Abusando de la notación y sobreentendiendo el contexto podemos afirmar que $\{53\}^{-1} = \{ca\}$.

4.4. AES y los Polinomios de Grado Tres

En la implementación de AES es necesario considerar la aritmética de $GF(2^8)[x]$ módulo el polinomio x^4+1, es decir, necesitamos conocer la aritmética de $GF(2^8)[x]/(x^4+1)$. No ofrece dificultad alguna entender que si $a(x), b(x) \in GF(2^8)[x]$, donde $a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ y $b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$, entonces:

$$a(x)+b(x) = (a_3\oplus b_3)x^3+(a_2\oplus b_2)x^2+(a_1\oplus b_1)x+(a_0\oplus b_0)$$

Pues cada a_i (resp. b_i) puede ser entendido como un byte. Debemos centrarnos ahora en el producto $a(x)b(x)$, que las explicaciones de AES representan por $a(x)\otimes b(x)$.

Teorema 4.4.1. *Para todo* $a(x), b(x) \in \mathrm{GF}(2^8)[x]$, *si* $a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$, $b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$ *y* $a(x) \otimes b(x) = a(x)b(x) \bmod (x^4 + 1)$ *es el polinomio* $d_3x^3 + d_2x^2 + d_1x + d_0$, *entonces:*

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{4.2}$$

Demostración. Si llevamos a cabo la multiplicación de $a(x)b(x)$ obtendremos el polinomio $c(x)$:

$$c(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0$$

donde:

$$\begin{aligned}
c_0 &= a_0 \bullet b_0 \\
c_1 &= a_1 \bullet b_0 \oplus a_0 \bullet b_1 \\
c_2 &= a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 \\
c_3 &= a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3 \\
c_4 &= a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3 \\
c_5 &= a_3 \bullet b_2 \oplus a_2 \bullet b_3 \\
c_6 &= a_3 \bullet b_3
\end{aligned}$$

Pero al ser $x^4 + 1 \equiv 0 \pmod{x^4 + 1}$ tenemos equivalentemente que $x^4 \equiv 1 \pmod{x^4 + 1}$; así pues, para todo número natural k,

$$\begin{aligned}
x^k \bmod (x^4 + 1) &= x^{(k \text{ div } 4)4 + (k \bmod 4)} \bmod (x^4 + 1) \\
&= x^{(k \text{ div } 4)4} x^{(k \bmod 4)} \bmod (x^4 + 1) \\
&= (x^{(k \text{ div } 4)4} \bmod (x^4 + 1))(x^{k \bmod 4} \bmod (x^4 + 1)) \\
&= (x^4 \bmod (x^4 + 1))^{k \text{ div } 4} (x^{k \bmod 4} \bmod (x^4 + 1)) \\
&= 1^{(k \text{ div } 4)} (x^{(k \bmod 4)} \bmod (x^4 + 1)) \\
&= x^{k \bmod 4} \bmod (x^4 + 1) \\
&= x^{k \bmod 4}
\end{aligned}$$

De ello se deduce que:

$$\begin{aligned}
d_0 &= a_0 \bullet b_0 \oplus a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3 \\
d_1 &= a_1 \bullet b_0 \oplus a_0 \bullet b_1 \oplus a_3 \bullet b_2 \oplus a_2 \bullet b_3 \\
d_2 &= a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 \oplus a_3 \bullet b_3 \\
d_3 &= a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3
\end{aligned}$$

que escrito en forma matricial es:

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

□

Observación 4.4.1. Dado que $x^4+1 = (x+1)^4$ como elemento de $GF(2^8)[x]$, dicho polinomio no es irreducible. Así pues si fijásemos un polinomio de $GF(2^8)[x]$ para multiplicar módulo x^4+1 y éste tuviese grado menor que 4, el producto no sería necesariamente reversible dado que dicho polinomio podría no tener inverso módulo x^4+1. No obstante, el cifrador AES ha fijado un polinomio que sí tiene inverso módulo x^4+1.

Teorema 4.4.2. *Sea el polinomio* $a(x) \in GF(2^8)[x]$ *definido por:*

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$

Existe $a(x)^{-1}$ mód (x^4+1) *y*

$$a(x)^{-1} \text{ mód } (x^4+1) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$$

Demostración. Aplicando en $GF(2^8)[x]$ el algoritmo de Euclides extendido a los polinomios $a(x)$ y x^4+1 obtenemos que:

$$\begin{aligned} 1 &= (a(x), x^4+1) \\ &= (\{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\})a(x) + (\{00\}x^3 + \{1d\}x^2 + \{1c\}x + \{1d\})(x^4+1) \end{aligned}$$

por lo que existe $a(x)^{-1}$ mód (x^4+1) y

$$a(x)^{-1} \text{ mód } (x^4+1) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$$

□

Observación 4.4.2. Otro polinomio usado en AES es:

$$a(x) = \{01\}x^3 + \{00\}x^2 + \{00\}x + \{00\}$$

por lo que si suponemos fijo dicho polinomio y variable $b(x)$ la transformación de (4.4) que calcula $a(x) \otimes b(x)$ es:

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

lo que supone una rotación de bytes en la palabra de entrada.

4.5. Especificaciones del Algoritmo

En cuanto al algoritmo AES, la longitud de los bloques de entrada y el Estado es 128 bits. Esto viene representado por $Nb = 4$, que responde a al número de palabras de 32 bits (número de columnas) en el Estado.

La longitud de la clave de cifrado, K, es: 128, 192 o 256 bits y dicha longitud es representada por Nk. Por tanto, Nk puede tomar los valores: 4, 6 u 8, repondiendo al número de palabras de 32 bits (número de columnas) en Clave de Cifrado.

En el algoritmo de AES, el número de rondas completadas durante la ejecución, representado por Nr, depende del tamaño de la clave. La relación está explicada en la tabla de la Figura 4.2.

modalidad	Longitud de la Clave Nk palabras	Tamaño del Bloque Nb palabras	Número de Rondas Nr
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Figura 4.2: Número de Rondas en AES.

Tanto para el cifrador como para el descifrador, el algoritmo de AES usa una función de ronda compuesta de cuatro transformaciones orientadas a bytes:

1. Sustitución de bytes según una tabla de sustitución o S-caja (S-box),
2. desplazamiento de filas del Estado según diferentes compensaciones,
3. mezcla de datos dentro de cada columna del Estado y
4. suma al Estado de una Clave de Ronda (Round Key).

4.6. Cifrador AES

A modo de esbozo, el cifrador parte del paquete de 128 bits ajustado a un esquema de 16 bytes y construye el Estado inicial (cfr. Sección 4.2). Tras una suma inicial con la clave de ronda, el estado inicial es transformado aplicando una función de ronda 10, 12 ó 14 veces, dependiendo de la longitud de la clave, donde la ronda final es ligeramente diferente de las primeras $Nr - 1$ rondas. El estado final es formateado para dar la salida (cfr. Sección 4.2).

El algoritmo de cifrado está detallado en la Figura 4.3 y sus partes en las siguientes subsecciones.

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])

   for round = 1 step 1 to Nr-1
      SubBytes(state)
      ShiftRows(state)
      MixColumns(state)
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for
   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   out = state
end
```

Figura 4.3: Pseudocódigo del Algoritmo de Cifrado de AES.

La transformación `SubBytes()`

La transformación `SubBytes()` es una sustitución no lineal de bytes que opera independientemente sobre cada byte del estado, s_{ij}, cambiándolo por s'_{ij}. Para encontrar el byte s'_{ij} asociado al s_{ij} utiliza una tabla de sustitución llamada S-box, que es la que aparece en la Figura 4.4.

La construcción de la tabla es como sigue. Para cada entrada xy:

1. Si el byte xy es distinto de 00, tomamos el inverso en $GF(2^8)$ de xy que es su aplicado; el byte 00 se aplica en él mismo.

2. Si b_i, donde $0 \leq i < 8$, es el i-ésimo bit del byte entonces su transformado b'_i se

calcula mediante la fórmula con operaciones en GF(2):

$$b'_i = b_i + b_{(i+4) \bmod 8} + b_{(i+5) \bmod 8} + b_{(i+6) \bmod 8} + b_{(i+7) \bmod 8} + c_i \tag{4.3}$$

donde para todo $0 \leq i < 8$, c_i es el i-ésimo bit del byte $\{63\}$ ó $\{01100011\}$.

Teorema 4.6.1. *Para todo byte* $b = \{b_7b_6b_5b_4b_3b_2b_1b_0\}$, *la transformación de (4.3) que lo convierte en* b' *tiene la siguiente expresión matricial:*

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Ejemplo 4.6.1. Consideremos el caso del byte $s = \{53\}$ del que queremos conocer su transformado por `SubBytes()`. A $\{53\}$ ó $\{01010011\}$ le corresponde el polinomio $n(x) = x^6 + x^4 + x + 1$ y como quiera que:

$$(x^7 + x^6 + x^3 + x)n(x) + (x^5 + x^4 + x^3 + x^2 + 1)m(x) = 1$$

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figura 4.4: S-box: valores de sustitución para el byte xy (en formato hexadecimal).

donde $m(x)$ es el polinomio de la Definición 4.3.1, se tiene que:

$$(x^7 + x^6 + x^3 + x)n(x) \equiv 1 \pmod{m(x)}$$

lo que nos lleva a saber que el inverso de $n(x)$ en el cuerpo $GF(2^8)$, tal y como lo hemos representado, es $x^7+x^6+x^3+x$ que es representado por el byte $\{11001010\}$. Ahora bien:

$$\begin{bmatrix}1\\1\\1\\0\\1\\1\\0\\1\end{bmatrix} = \begin{bmatrix}1&0&0&0&1&1&1&1\\1&1&0&0&0&1&1&1\\1&1&1&0&0&0&1&1\\1&1&1&1&0&0&0&1\\1&1&1&1&1&0&0&0\\0&1&1&1&1&1&0&0\\0&0&1&1&1&1&1&0\\0&0&0&1&1&1&1&1\end{bmatrix}\begin{bmatrix}0\\1\\0\\1\\0\\0\\1\\1\end{bmatrix} + \begin{bmatrix}1\\1\\0\\0\\0\\1\\1\\0\end{bmatrix}$$

y $\{11101101\}$ es ed. Por eso la posición 53 en la Tabla 4.4 viene ocupada por ed; en resumen, si 53 fuese un byte del estado entonces su transformado al actuar `SubBytes()` sobre ese estado sería ed.

La transformación `ShiftRows()`

La transformación `ShiftRows()` toma un estato S y actúa sobre él produciendo el estado `S.ShiftRows()`, al que llamamos `S'`; la actuación es fila a fila mediante una rotación en cada una, pero dejando intacta la primera. En efecto, para todo $0 \leq r < 4$ la fila $[s_{r0}, s_{r1}, s_{r2}, s_{r3}]$ del estado es sustituida por $[s'_{r0}, s'_{r1}, s'_{r2}, s'_{r3}]$ según la siguiente regla:

$$s'_{rc} = s_{r((c+\text{shift}(r,Nb)) \bmod Nb)}, \text{ para todo } 0 \leq r < 4 \text{ y } 0 \leq c < Nb$$

donde para todo $0 \leq r < 4$, $\text{shift}(r, Nb) = r$ (sea recordado que $Nb = 4$). Esquemáticamente el resultado de aplicar `ShiftRows()` sobre el estado es según indica la Figura 4.5.

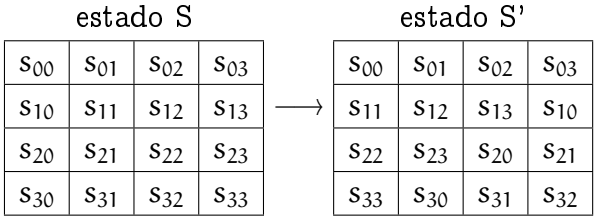


Figura 4.5: Esquema del efecto de aplicar `ShiftRows()` a un estado S.

La transformación `MixColumns()`

La transformación `MixColumns()` actua sobre el estado columna a columna, considerando cada columna como un polinomio de $GF(2^8)[x]$ de grado 3. Esos polinomios son multiplicados módulo x^4+1 por

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$

como ha sido detallado en la Sección 4.4. En efecto, si $s(x)$ representa a una columna del estado, su tranformada como parte de `MixColumns()` es $s'(x) = a(x) \otimes s(x)$ y esto puede ser llevado a cabo por medio de una multiplicación matricial.

Corolario 4.6.2. *Sea S un estado y $s(x)$ el polinomio de $GF(2^8)[x]$ que representa a una cualquiera de sus columnas. Entonces, para $s'(x) = a(x) \otimes s(x)$ se cumple:*

$$\begin{aligned}
s'_{0c} &= (\{02\} \bullet s_{0c}) \oplus (\{03\} \bullet s_{1c}) \oplus s_{2c} \oplus s_{3c} \\
s'_{1c} &= s_{0c} \oplus (\{02\} \bullet s_{1c}) \oplus (\{03\} \bullet s_{2c}) \oplus s_{3c} \\
s'_{2c} &= s_{0c} \oplus s_{1c} \oplus (\{02\} \bullet s_{2c}) \oplus (\{03\} \bullet s_{3c}) \\
s'_{3c} &= (\{03\} \bullet s_{0c}) \oplus s_{1c} \oplus s_{2c} \oplus (\{02\} \bullet s_{3c})
\end{aligned}$$

Demostración. Dado que $s'(x)$ ha sido definido como $a(x) \otimes s(x)$ y que $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$, aplicamos lo que establece el Teorema 4.4.1 y obtenemos que:

$$\begin{bmatrix} s'_{0c} \\ s'_{1c} \\ s'_{2c} \\ s'_{3c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0c} \\ s_{1c} \\ s_{2c} \\ s_{3c} \end{bmatrix}$$

y de ahí el resultado. □

Expansión de la Clave

El algoritmo de AES toma la *clave de cifrado* (cipher key), K y le aplica una rutina de *expansión de clave* (key expansion) para generar una *clave extendida* (key schedule); la expansión de la clave genera un total de $Nb(Nr+1)$ palabras. El algoritmo requiere un conjunto inicial de Nb palabras y cada una de las Nr rondas requiere Nb palabras de clave dato. La clave extendida resultante es un array lineal de palabras de 4 bytes denotado por `w[i]`, donde $0 \leq i < Nb(Nr+1)$. La expansión de la clave entrada (input) hasta obtener la clave extendida (key schedule) es según el pseudocódigo de la Figura 4.6

Las funciones básicas del proceso son:

- `Rcon[i]`, array de palabras constante para la ronda `i`, es $[x^{i-1}, \{00\}, \{00\}, \{00\}]$ donde x^{i-1} es la potencia de x (denotado por $\{02\}$) en el cuerpo $GF(2^8)$ (nótese que `i` comienza en 1 y no en 0).
- `SubWord()`; es una función que toma una palabra de entrada de cuatro bytes y le aplica la sustitución de la S-box (cfr. Figura 4.4) a cada uno de los cuatro bytes para producir la palabra de salida.
- `RotWord()`; es una función que toma la palabra $[a_0, a_1, a_2, a_3]$ como entrada, lleva a cabo una permutación cíclica y devuelve la palabra $[a_1, a_2, a_3, a_0]$.

y el proceso de expansión de clave es llevado a cabo por el procedimiento `KeyExpansion()` (cfr. Figura 4.6) es como sigue.

Los argumentos de `KeyExpansion()` son: el parámetro Nk (cfr. la tabla de la Figura 4.2) y una lista `key` de bytes de longitud 4Nk y la salida una lista `w` de palabras (word) de longitud $Nb(Nr+1)$. El algoritmo por etapas procede como sigue:

- Toma la clave `key` de 4Nk bytes y compone una lista `w` de palabras (word) de 4 bytes o 32 bits y de longitud Nk. La construcción de de `w` es agrupando bytes adjuntos de `key`.
- La variable `i` toma entonces el valor Nk y servirá de centinela en un bucle `while` en el que se usa una variable temporal `temp` que toma como valores palabras (word).
- La condición de parada del bucle es que `i` llegue a tomar el valor $Nb(1+Nr)$; en el mismo se incrementa el valor de `i` en una unidad por vuelta.
- En el bucle `temp` empieza valiendo `w[i-1]`. Así pues en la primera vuelta `temp` toma el valor de la última palabra de `w`. El valor de `temp` es eventualmente modificado según el valor de `i` y ello mediante el uso de: `SubWord`, `RotWord` y los valores de `Rcon`.
- Finalmente se añade a `w` en la ronda `i` el resultado de operar `temp` y `w[i-Nk]` mediante `xor`.
- Al acabar el bucle, el resultado `w` es aportado como resultado del procedimiento.

En las descripciones clásicas del algoritmo `w` es llamada *clave extendida* (key schedule). Se le llama *clave de (la) ronda* (round key) `k` a `[w[kNb+c]:` $0 \leq c < Nb$`]`, donde $1 \leq k < Nr$.

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
   word temp

   i=0

   while (i < Nk)
      w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
      i = i+1
   end while

   i = Nk

   while (i < Nb * (Nr+1)]
      temp = w[i-1]
      if (i mod Nk = 0)
         temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
      else if (Nk > 6 and i mod Nk = 4)
         temp = SubWord(temp)
      end if
      w[i] = w[i-Nk] xor temp
      i=i+1
   end while
end

Note that Nk=4, 6, and 8 do not all have to be implemented;
they are all included in the conditional statement above for
conciseness. El algoritmo soporta llaves de 128, 192 ó 256 bits
(i.e. Nk = 4, 6 ó 8 respectivamente).
```

Figura 4.6: Pseudocódigo del Algoritmo de Expansión de Clave.

Para ver ejemplos de expansión de claves puede consultar el documento NIST.FIPS.197.pdf.

La transformación `AddRoundKey()`

La transformación `AddRoundKey()` consiste en operar una clave de ronda con el estado mediante una simple `xor`. `AddRoundKey()` transforma el estado S en el S' columna a columna según la relación:

$$[s'_{0c}, s'_{1c}, s'_{2c}, s'_{3c}] = [s_{0c}, s_{1c}, s_{2c}, s_{3c}] \oplus w[kNb + c], \qquad 0 \leq c < Nb$$

donde k representa a la ronda en la que estamos siendo $1 \leq k < Nr$.[2]

4.7. Cifrador AES Inverso

La operaciones de cifrado que han sido explicadas en la Sección 4.6 pueden ser invertidas, dando lugar al cifrador inverso de AES o *algoritmo de descifrado de AES*. La explicación de este algoritmo en pseudocódigo es la dada en la Figura 4.7; las funciones que aparecen involucradas serán explicadas en las subsecciónes siguientes.

La transformación `InvShiftRows()`

La transformación `InvShiftRows()` toma un estato S y actúa sobre él produciendo el estado S'; la actuación es fila a fila mediante una rotación en cada una, pero dejando intacta la primera. En efecto, para todo $0 \leq r < 4$ la fila $[s_{r0}, s_{r1}, s_{r2}, s_{r3}]$ del estado es sustituida por $[s'_{r0}, s'_{r1}, s'_{r2}, s'_{r3}]$ según la siguiente regla:

$$s'_{r((c+shift(r,Nb)) \text{ mód } Nb)} = s_{rc}, \text{ para todo } 0 \leq r < 4 \text{ y } 0 \leq c < Nb$$

donde para todo $0 \leq r < 4$, $shift(r, Nb) = r$ (sea recordado que $Nb = 4$). Esquemáticamente el resultado de aplicar `InvShiftRows()` sobre el estado es según indica la Figura 4.8.

Transformación `InvSubBytes()`

La transformación `InvSubBytes()` es la inversa de `SubBytes()`. Es obtenida por aplicación de la inversa de transformación afín (4.3) seguida de la toma del inverso multipli-

[2] Aquí tanto $[s_{0c}, s_{1c}, s_{2c}, s_{3c}]$ como w[j] deben ser consideradas como una palabras (de 32 bits).

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
  byte state[4,Nb]

  state = in

  AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
  for round = Nr-1 step -1 downto 1
     InvShiftRows(state)
     InvSubBytes(state)
     AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
     InvMixColumns(state)
  end for

  InvShiftRows(state)
  InvSubBytes(state)
  AddRoundKey(state, w[0, Nb-1])

  out = state
end
```

Figura 4.7: Pseudocódigo del Algoritmo de Cifrado Inverso de AES.

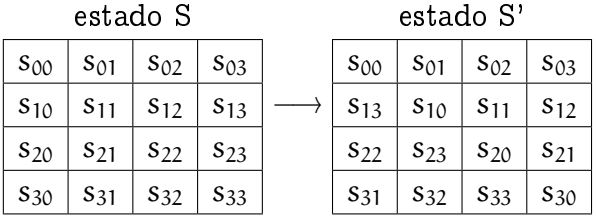


Figura 4.8: Esquema del efecto de aplicar `InvShiftRows()` a un estado S.

cativo en $GF(2^8)$. La S-box inversa usada en la transformación `InvSubBytes()` es la de la Figura 4.9.

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Figura 4.9: S-box inversa: valores de sustitución para el byte xy (en formato hexadecimal).

Transformación `InvMixColumns()`

La transformación `InvMixColumns()` es la inversa de la transformación `MixColumns()`. `InvMixColumns()` opera en el estado columna a columna, tratando cada columna como un polinomio de grado 3 como ha sido detallado en la Sección 4.4. Las columnas son consideradas como polinomios sobre $GF(2^8)$ y multiplicadas módulo $x^4 + 1$ con un polinomio fijo $a(x)^{-1}$; dicho polinomio inverso existe, según afirma el Teorema 4.4.2, y vale:

$$a(x)^{-1} \text{ mód } (x^4 + 1) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$$

Como corolario del Teorema 4.4.1 tenemos el siguiente:

Corolario 4.7.1. *Para todo* $s(x) \in GF(2^8)[x]$, *si* $s(x) = s_3x^3 + s_2x^2 + bs_1x + s_0$ *y*

$a(x)^{-1} \otimes s(x) = a(x)^{-1} s(x)$ mód $(x^4 + 1)$ *es el polinomio* $s'_3 x^3 + s'_2 x^2 + s'_1 x + s'_0$, *entonces:*

$$\begin{bmatrix} s'_0 \\ s'_1 \\ s'_2 \\ s'_3 \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} \tag{4.4}$$

Por tanto, a efectos prácticos, interpretaremos cada columna del estado S como un polinomio de GF(2^8), $s_c(x) = s_{3c} x^3 + s_{2c} x^2 + b s_{1c} x + s_{0c}$, con grado 3 y $0 \leq c < Nb$ para multiplicarlo por $a(x)^{-1}$ obteniendo lo siguiente:

$$\begin{bmatrix} s'_{0c} \\ s'_{1c} \\ s'_{2c} \\ s'_{3c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0c} \\ s_{1c} \\ s_{2c} \\ s_{3c} \end{bmatrix} \qquad 0 \leq c < Nb$$

que se traduce en:

$$\begin{aligned} s'_{0c} &= (\{0e\} \bullet s_{0c}) \oplus (\{0b\} \bullet s_{1c}) \oplus (\{0d\} \bullet s_{2c}) \oplus (\{09\} \bullet s_{3c}) \\ s'_{1c} &= (\{09\} \bullet s_{0c}) \oplus (\{0e\} \bullet s_{1c}) \oplus (\{0b\} \bullet s_{2c}) \oplus (\{0d\} \bullet s_{3c}) \\ s'_{2c} &= (\{0d\} \bullet s_{0c}) \oplus (\{09\} \bullet s_{1c}) \oplus (\{0e\} \bullet s_{2c}) \oplus (\{0b\} \bullet s_{3c}) \\ s'_{3c} &= (\{0b\} \bullet s_{0c}) \oplus (\{0d\} \bullet s_{1c}) \oplus (\{09\} \bullet s_{2c}) \oplus (\{0e\} \bullet s_{3c}) \end{aligned}$$

4.8. Cifrador AES Inverso Equivalente

En la Sección 4.7 fue presentado el obvio mecanismo de descifrado de AES y este quedó fijado en la Figura 4.7. Sin embargo la sucesión de transformaciones difieren de la del cifrador, mientras que la forma de la clave extendida para el cifrado y el descifrado son la misma.

No obstante, algunas propiedades del algoritmo de AES conducen a una versión equivalente del algoritmo de descifrado (cifrado inverso) que tiene la misma sucesión de transformaciones que el cifrador, pero con las transformaciones sustituidas por sus inversas. Esto es llevado a cabo con un cambio en la clave extendida. Las dos propiedades que conducen a este cifrador inverso equivalente son como siguen:

1. Las transformaciones `SubBytes()` y `ShiftRows()` conmutan; lo mismo ocurren con sus inversas `InvSubBytes()` e `InvShiftRows()`.

2. Las operaciones de mezclado de columnas: `MixColumns()` y `InvMixColumns()` tienen la siguiente propiedad de linealidad:

```
InvMixColumns(state XOR Round Key)=InvMixColumns(state) XOR
                    InvMixColumns(Round Key)
```

Estas propiedades permiten revertir el orden de `InvSubBytes()` y `InvShiftRows()`. También puede ser revertido el orden de las transformaciones `AddRoundKey()` y `InvMixColumns()` con tal de que las (palabras) columnas de la llave extendida para el descifrado sea modificada utilizando la transformación `InvMixColumns()`.

El descifrador, o cifrador inverso, equivalente es definido revirtiendo el orden de las transformaciones `InvSubBytes()` e `InvShiftRows()` respecto a como aparecen en el pseudocódigo de la Figura 4.7 y revirtiendo el orden de las transformaciones `AddRoundKey()` y `InvMixColumns()` utilizadas en el "bucle ronda" después de la primera modificación de la clave extendida de descifrado para las rondas de 1 a Nr-1 utilizando la transformación `InvMixColumns()`. La primera y la última Nb palabras de la clave extendida de descifrado *no* serán modificadas de esta forma.

Implementado estos cambios, el Cifrador Inverso Equivalente ofrece una estructura más eficiente que el Cifrador Inverso descrito en la Figura 4.7. El pseudocódigo para el Cifrador Inverso Equivalente aparece en la Figura 4.10. El array de palabras `dw[]` contiene la modificada clave extendida de descifrado. La modificación de la rutina para la expansión de la clave está incluida también en la Figura 4.10.

```
EqInvCipher(byte in[4*Nb], byte out[4*Nb], word dw[Nb*(Nr+1)])
begin
   byte state[4,Nb]

   state = in

   AddRoundKey(state, dw[Nr*Nb, (Nr+1)*Nb-1])

   for round = Nr-1 step -1 downto 1
      InvSubBytes(state)
      InvShiftRows(state)
      InvMixColumns(state)
      AddRoundKey(state, dw[round*Nb, (round+1)*Nb-1])
   end for

   InvSubBytes(state)
   InvShiftRows(state)
   AddRoundKey(state, dw[0, Nb-1])

   out = state
end

For the Equivalent Inverse Cipher, the following pseudo code is added
at  the end of the Key Expansion routine:

   for i = 0 step 1 to (Nr+1)*Nb-1
      dw[i] = w[i]
   end for

   for round = 1 step 1 to Nr-1
      InvMixColumns(dw[round*Nb, (round+1)*Nb-1]) // note change
                                                  // of type
end for

Note that, since InvMixColumns operates on a two-dimensional array of
bytes while the Round Keys are held in an array of words, the call to
InvMixColumns in this code sequence involves a change of type (i.e.
the input to InvMixColumns() is normally the State array, which is
considered to be a two-dimensional array of bytes, whereas the input
here is a Round Key computed as a one-dimensional array of words).
```

Figura 4.10: Pseudocódigo del Algoritmo de Cifrado Inverso Equivalente de AES.

Capítulo 5

Criptografía de Clave Pública

5.1. Generalidades

En los criptosistemas considerados hasta ahora, emisor y receptor selecciona una clave K para el algoritmo de cifrado E_K. Esto determina el algoritmo de descifrado D_K. Como la misma clave es utilizada en ambas operaciones, las dos partes deben tomar gran cuidado en el intercambio de la clave de forma que un espía no la obtenga. El problema del intercambio de claves es uno de los más difíciles en los criptosistemas clásicos. No es posible usar el teléfono o un sistema de correo para el intercambio de claves en tanto que los canales son inseguros. El emisario es habitualmente el medio más seguro y fiable para intercambiar claves. En el caso de las comunicaciones militares y diplomáticas es posible usar emisarios, pero es muy costoso y poco práctico en el ámbito comercial. En una situación comercial las transacciones son establecidas entre dos o más partes que podrían incluso no conocerse entre sí y que nunca volverán a hacer negocios juntos, pero que no obstante desean plena confidencialidad y comunicación instantánea segura.

Existen además otros problemas para la administración de claves en los criptosistemas clásicos. Considere una red de usuarios que desean comunicarse entre sí, por ejemplo, una red de bancos o un grupo de gente intercambiando mensajes electrónicos. Para tener una completa seguridad, cada par de usuarios debe contar con una clave diferente a la de las demás. Por ejemplo, con 10 usuarios necesitamos 45 claves diferentes y con 100 usuarios necesitaríamos 4950 claves. No es fácil añadir un nuevo usario a la red, en tanto que la base de datos de usuarios necesitaría ser actualizada hasta incluirlo. Por otra parte, el gran número de claves genera problemas de seguridad al ser necesario que cada usuario salvaguarde su propia base. Imaginemos además el problema logístico que supondría la necesaria renovación de claves para el círculo con el trascurso del tiempo.

Los criptosistemas clásicos adolecen de problemas de crédito (problems of trust) ya que ambos usuarios tienen la misma clave en cada pareja emisor/receptor. Suponga que Alice y Bob comparten una clave secreta. Si Carol obtuviera la clave, ella podría suplantar a Bob enviando mensajes a Alice, quien no tiene ninguna forma de conocer que el mensaje que ella ha recibido no proviene de Bob. Carol podría interceptar mensajes de Alice a Bob y modificarlos en el tránsito sin que Bob se percate de ello. Es también posible que Bob falsifique mensajes, es decir, que envie un mensaje y más tarde niegue que él haya enviado ese mensaje alegando que su clave ha sido robada. El problema de la falsificación no es serio en el ámbito militar/diplomático, donde es más probable que las partes hayan establecido confianza, pero es una de las principales preocupaciones en situaciones comerciales. Un cliente de un banco no debería poder negar que ha hecho una transacción, pero al mismo tiempo el cliente debería poder estar seguro de que el banco no está falsificando transacciones.

Necesitamos que un criptosistema satisfaga los siguientes condiciones, además de ser seguro criptoanalíticamente:

- *Autenticación*: el receptor del mensaje debería ser capaz de cersiorarse de su origen.
- *Integridad*: el receptor de un mensaje debería ser capaz de determinar que el mensaje no ha sido modificado en el tránsito.
- *No repudiabilidad*: el emisario de un mensaje debería de ser incapaz de negar posteriormente que el envió el mensaje.

Estas condiciones son imposibles de alcanzar en los criptosistemas clásicos de clave única. El progreso vino en 1976 con la invención por Diffie y Hellman, e independientemente por Merkle, de la Criptografía de Clave Pública.

La criptografía de clave pública fue inventada para el manejo de claves secretas. En un tal sistema no hay necesidad de intercambiar claves. Cada usuario usa dos claves, una clave de cifrado y una clave de descifrado. La clave de cifrado es pública (de ahí el nombre de "clave pública") mientras que la clave de descifrado es mantenida secreta. Cualquiera que deseara mandar un mensaje a una persona usaría la clave pública de esa persona para cifrar el mensaje. El punto aquí es que el conocimiento del método de cifrado no permite el descifrado, asegurando que no es necesario el intercambio de claves secretas. Se puede enviar información confidencial mediante las claves publicadas confiando en que sólo el destinatario podrá descifrarlo.

Podría ayudar al lector el considerar una analogía con el sistema de correo. Podemos enviar correos a cualquiera siempre que conozcamos tan sólo su dirección y una vez que

esté en el buzón del receptor, sólo el receptor puede leerlo (suponiendo que el buzón solamente puede ser abierto con la llave del receptor). El sistema de clave pública es más fuerte en el sentido de que cada mensaje viaja en su propia caja segura, la cual puede ser abierta únicamente por el propietario legítimo. Incluso ni el emisor podría abrir la caja para revelar su contenido.

En un criptosistema de clave pública cada usuario tiene dos claves. Esto reduce en gran medida el número de claves necesarias para establecer una red de comunicación. Cada usuario tiene una clave pública, la cual proporciona una función de cifrado E, y una clave privada, la cual proporciona una función de descifrado D. Para que el sistema funcione correctamente debemos tener para cualquier mensaje m:

$$D(E(m)) = m$$

La función de cifrado E es conocida para cualquiera. Como E es conocida, un oponente puede generar cualquier cantidad de texto cifrado. Por consiguiente, cualquier tal sistema debería ser inmune a un ataque basado en texto cifrado en el cual el correspondiente texto llano es también conocido. Para que el sistema sea práctico, se debería ser capaces de calcular $E(m)$ y $D(E(m))$ rápidamente, pero por seguridad, debería ser imposible determinar D a partir de E en un tiempo razonable. Es más fácil describir los requisitos de un sistema de clave pública que construir las funciones E y D que cumplan estos requisitos.

Teóricamente podría ser construido un sistema de clave pública como caso especial de una *función unidireccional* conocida como *función unidireccional trampa*. Intuitivamente una función unidireccional f debería ser biyectiva y $f(x)$ fácil de calcular para cualquier entrada x, pero f^{-1} debe ser difícil de calcular. Eso significa que sería difícil resolver la ecuación $f(x) = y$ cuando y es conocido. Una función unidireccional trampa es una función para la que la ecuación $f(x) = y$ resulta fácil de resolver si hay disponible información adicional (la trampa). En lugar de dar una definición precisa de qué significa "fácil de calcular y difícil de calcular" consideraremos un par de ejemplos. Subrayaremos que nadie ha demostrado la existencia de una tal función unidireccional. Las siguientes dos funciones son consideradas las más probables candidatas a ser función unidireccional.

Ejemplo 5.1.1. Fijemos dos enteros g y n y consideremos $f(x) = g^x \bmod n$. La función f es considerada fácil de calcular porque el número de pasos necesarios para evaluar f es una función polinomial del número de dígitos en las entradas: x, g y n. La exponencial puede ser calculada en a los sumo $\log_2(x)$ operaciones de cuadrado. El cuadrado de y^2 de un número y puede ser calculado en $(\log_2(y))^2$ pasos. En cualquier caso, el número de pasos es una función polinomial del tamaño de la entrada, el número de dígitos. El problema de calcular x a partir del conocimiento de $g^x \bmod n$, g y n es conocido como el *problema del logaritmo discreto*. A pesar de muchas investigaciones al respecto,

no es conocido ningún método para evaluar el logaritmo discreto general en tiempo polinomial en el número de dígitos. Los algoritmos conocidos requieren tiempo que es polinomial en la entrada $g^x \bmod n$, no polinomial en el tamaño de la entrada. Para resolver esta pregunta, se necesita demostrar que el problema del logaritmo discreto es difícil o encontrar un algoritmo que lo calcule en un tiempo polinomial en el tamaño de la entrada.

Ejemplo 5.1.2. Considere la función $f(p, q) = pq$, donde p y q son números primos. La función es fácil de calcular porque la multiplicación toma tiempo proporcional al producto de número de dígitos de p y de q, pero calcular la inversa de f es un problema extremadamente difícil a día de hoy, aunque haya tests de primalidad en tiempo polinomial que si fallan no dan idea de la factorización. No hay ningún método eficiente de determinar p y q a partir de $n = pq$. El método de factorizar probando a buscar divisores toma tiempo proporcional a $\sqrt{p}$, si p es el menor de los dos primos. Esto es exponencial en el número de dígitos. Hay métodos modernos de factorización que no puede factorizar números con más de 150 dígitos. Si p y q tienen cada uno 130 dígitos, a los efectos prácticos tenemos una función unidireccional. No obstante, no hay ninguna demostración de que estemos ante una verdadera función unidireccional.

La primera aplicación de la función unidireccional parece haber sido la protección de contraseñas de usuarios en sistemas multiusuario. Un usuario necesita que la contraseña para poder acceder al sistema sea conocida sólo por él. Si la computadora almacena las contraseñas en un fichero, entonces la secrecidad de las contraseñas es susceptible de ataque. La solución es aplicar una función unidireccional a la contraseña y almacenar los resultados en el fichero de contraseñas. Para autenticar a un usuario, la computadora aplica la función unidireccional a la contraseña tecleada y compara el resultado con la entrada conrrespondiente en el fichero de contraseñas. Si hay conincidencia entre ambos valores, al usuario le es permitido el acceso. La ventaja de este sistema es que la contraseña real no es almacenada y, si la función usada es unidireccional, entonces cualquiera que vea las entradas de contraseña será incapaz de determinar la contraseña real.

A diferencia del ejemplo anterior, para cifrar y descifrar, una función unidireccional no es directamente útil. Lo que se precisa es un tipo especial de función de una dirección, la función unidireccional trampa. En el Ejemplo 5.1.2, si $n = pq$ es conocido entonces p y q pueden ser determinados si una información adicional está disponible, por ejemplo $\phi(n)$ o $\sigma(n)$. Esta información adicional es la trampa. Si esas funciones son usadas en criptografía, para determinar el algoritmo de descifrado a partir del conocimiento de E la trampa es necesaria. Los siguientes son los sistemas de clave pública que han sido propuestos:

- *RSA*; este método es descrito en la Sección 5.2. Su seguridad radica en la dificultad

del problema de factorización de enteros.

- *ElGamal*; este método está basado en la presunta intratabilidad del problema del logaritmo discreto.
- *McEliece*; el sistema McEliece está basado en la teoría de los códigos Goppa. La teoría no está en nuestro ámbito.
- *Curvas Elípticas*; los criptosistemas de curvas elípticas son generalización del sistema ElGamal y están basados en la dificultad de calcular logaritmos discretos en curvas elípticas.

La primera aplicación de los sistemas de clave pública es al problema de intercambio de llaves en sistemas de clave secreta. El siguiente protocolo fue propuesto por Diffie y Hellman.[1] Supongase que Alice y Bob desean intercambiar una clave secreta:

1. Los usuarios Alice y Bob eligen conjuntamente dos enteros elevados, n y g, tales que $1 < g < n$. La pareja $\langle n, g\rangle$ no tiene que por qué ser secreta y puede ser compartida por más usuarios.
2. El usuario Alice elige aleatoriamente un número elevado x y calcula $X = g^x$ mód n.
3. El usuario Bob elige aleatoriamente un número elevado y y calcula $Y = g^y$ mód n.
4. Los usarios Alice y Bob se intercambian los valores X e Y manteniendo secretos los exponentes de los que se sirvieron para generarlos.
5. El usuario Alice calcula $k = Y^x$ mód n, es decir, calcula $k = (g^y)^x$ mód n.
6. El usuario Bob calcula $k' = X^y$ mód n, es decir, calcula $k = (g^x)^y$ mód n.

k es la clave secreta que calculan independientemente y que comparten los usuarios Alice y Bob. Este esquema puede ser extendido a más de dos usuarios.

Observación 5.1.1. Sea observado que $k = k' = g^{xy}$ mód n y que aunque sean conocidos n, g, X e Y, no será posible encontrar los valores x o y a menos que se sepa calcular el logaritmo discreto —lo cual no es generalmente fácil. La clave k se utilizará ahora de la forma adecuada:

1. expresándola en base 2 y tomando la cadena de bits que se considere necesaria para cifrar con AES.

[1] W. Diffie and M.E. Hellman. "New Directions in Cryptography." *IEEE Transactions in Information Theory*, 22(1976), 644–654.

2. tomarla como clave de un sistema de cifrado clásico.

Es aconsejable que la elección de g y n esté sujeta a las siguientes condiciones, si se quiere alcanzar un adecuado grado de seguridad:

1. n debe ser un primo seguro, es decir, un número primo de la forma $2p+1$ donde p es primo (p.e. 23, 47 y 59).
2. El tamaño de n debe ser de 512 bits o incluso 1028.
3. g es un elemento primitivo del cuerpo $GF(n)$, o sea, un generador del grupo cíclico $GF(n)^*$.

Ejemplo 5.1.3. Son elegidos $n = 83$ y $g = 2$. El número 83 es un primo seguro que puede ser expresado como $2 \cdot 41 + 1$. Por otra parte 2 es un elemento primitivo de $GF(83)$, dado que $2^2 = 4$ mód 83 y $2^{41} \equiv 82$ mód 83. Ambos valores son distintos de 1 en $GF(83)$. Tenemos la clave pública $\langle 83, 2\rangle$. Alice, la usuario A, escoge $x = 59$ y Bob, el usuario B, escoge $y = 41$. Tenemos que 2^{59} mód $83 = 53$ y 2^{41} mód $83 = 82$, de forma que Bob recibe de Alice el número 53 y Alice recibe de Bob el número 82. Ahora cada uno elevará el valor recibido a su propio número secreto y así obtener la clave compartida que no es otra que 82, ed decir, $k = k' = 82$.

El protocolo de intercambio de clave de Diffie-Hellman es susceptible de recibir un ataque por persona interpuesta. A Carol le sería posible interceptar X e Y y sustituirlos por sus X' e Y'. Si ella puede interceptar todas las comunicaciones entre Alice y Bob, entonces ella puede sustituir sus propios mensajes. Este tipo de ataque puede ser frustrado combinando un protocolo de intercambio de claves con esquemas de autenticación e identificación.

Una aplicación importante de los sistemas de clave pública es la autenticación de usuarios y la integridad de mensajes. Como simple ejemplo, supongase que un sistema de clave pública **satisface** $E(D(m)) = m$ además de satisfacer $D(E(m)) = m$ (lo último es obligatorio para un cifrado y descifrado exitosos). Denotaremos por E_A y E_B las claves públicas de cifrado de Alice y Bob, respectivamente, y D_A y D_B sus respectivas claves privadas. El protocolo básico de firma es como sigue:

1. Alice cifra el mensaje m con su clave privada D_A.
2. Alice envía $E_B(D_A(m))$ a Bob.
3. Bob recupera $D_A(m)$ con su clave privada y entonces recupera m por medio de la clave pública de Alice.

Al ser capaz Bob de recuperar m por medio de la clave pública de Alice, está seguro de que solamente Alice podría haberlo enviado dado que sólo ella conocería su clave privada. La seguridad del esquema deriva de la inviabilidad de calcular D_A a partir del conocimiento de E_A. La firma satisface las siguientes propiedades:

1. Bob puede verificar que Alice envía el mensaje porque ella lo firmó con su clave privada.

2. La firma depende del contenido del mensaje; por tanto, nadie puede usar la firma con otro documento.

En la práctica, en lugar de firmar el documento entero, es firmado sólamente un resumen (hash) del documento. La función resumen del documento es un valor que depende del documento. Usualmente es de un tamaño fijado, pero la función es tal que es difícil encontrar dos mensajes con el mismo resumen por ella. En el esquema de firma, la función resumen de un documento está firmada con la clave privada y puede ser verificada por cualquiera utilizando la clave pública. Más adelante serán discutidos con detalle los protocolos de firma.

La criptografía de clave pública ha encontrado numerosas aplicaciones:

1. Pueden ser autentificados documentos en papel tales como: cheques, valores, boletos de lotería, etc. A un documento le es asignado una única firma digital basada en el patrón de fibra del papel y el contenido del documento. La firma es cifrada con una clave privada y la cifra resultante es pegada (affixed to) en el documento. Cualquiera puede verificar la autenticidad del documento utilizando una clave pública incrustada.

2. La firma digital y los métodos de autenticación son usados en el control de acceso en redes. Por este método, una contraseña de usuario nunca necesitará viajar por la red.

3. Las técnicas de clave pública son actualmente usadas en tarjetas inteligentes, moneda digital y otros tipos de banca electrónica y comercio.

4. La verificación de algunos tratados internacionales puede hacerse mediante técnicas de clave pública.

5.2. El Criptosistema RSA

Esquemáticamente el cifrado de Pohlig-Hellman es una manipulación aritmética basada en un número primo p y un exponente e que funge ahora como clave de cifrado. El cifrado es llevado a cabo mediante la función $E(m)$ definida como sigue:

$$E(m) = m^e \pmod{p}$$

La función de descifrado es $D(m)$, en la que interviene una "clave de descifrado" d elegida de tal forma que se cumpla $ed \equiv 1 \pmod{(p-1)}$ y entonces

$$D(m) = m^d \pmod{p}$$

El método de *cifrado RSA* es una simple modificación del método de Pohlig-Hellman con el fin de hacer muy difícil la determinación de d. La idea es usar como módulo n el producto de dos números primos pq.

Definición 5.2.1. Sean:

1. p y q, dos números primos distintos.
2. n, el producto de p y q (i.e. $n = pq$).
3. $\phi(n)$, donde ϕ es la función totient de Euler.
4. e tal que $(e, \phi(n)) = 1$
5. d tal que $ed \equiv 1 \pmod{\phi(n)}$, es decir, $d = e^{-1} \bmod \phi(n)$.

La upla $\langle p, q, e\rangle$ es denominada *juego de clave RSA* y representada por K. La función de cifrado, $E_K(m)$, es la definida por:

$$E_K(m) = m^e \pmod{n}$$

y la de descifrado, $D_K(y)$, es la definida por:

$$D_K(y) = y^d \pmod{n}$$

Observación 5.2.1. Cada sujeto, para recibir documentación cifrada en un círculo RSA, incluirá en un listín público la pareja $\langle n, e\rangle$ y guardará celosamente cualquiera de las entradas de la 4-upla $\langle p, q, d, \phi(n)\rangle$.

Lema 5.2.1. *Si* $K = \langle p, q, e\rangle$ *es un juego de clave RSA y* $d = e^{-1} \bmod \phi(n)$*, entonces* $K' = \langle p, q, d\rangle$ *también lo es y* $D_K = E_{K'}$*.*

Es necesario demostrar ahora que las funciones de cifrado y descifrado son realmente operaciones inversas.

Teorema 5.2.2. *Sea* $K = \langle p, q, e\rangle$ *un juego de clave RSA y tanto* E_K *como* D_K *las funciones de la Definición 5.2.1 correspondientes a* K*. Entonces* $D_K \circ E_K = I_n$*, esto es, para todo número natural* m *tal que* $0 \leq m < n$*:*

$$D_K(E_K(m)) = m$$

Demostración. Tenemos que demostrar que para todo número m tal que $0 \leq m < n$,

$$m^{ed} \equiv m \pmod{n}$$

donde $ed \equiv 1 \pmod{\phi(n)}$, es decir, $\phi(n) \mid ed-1$. Dado cualquier m tal que $0 \leq m < n$, estaremos en uno de los siguientes dos casos:

1. $(m, n) = 1$; por el *teorema de Euler* (cfr. Teorema B.1.3) sabemos que:

   $$m^{ed-1} \equiv 1 \pmod{n} \tag{5.1}$$

   y multiplicando por m ambos miembros de la congruencia (5.1) sabemos que ésta se mantiene, por lo que:

   $$m^{ed} \equiv m \pmod{n} \tag{5.2}$$

2. $(m, n) \neq 1$; para fijar ideas supongamos que $m = pk$ tal que $(k, q) = 1$. Por una parte y dado que $p \equiv 0 \pmod{p}$,

   $$\begin{aligned} m^{ed} &\equiv 0 \pmod{p} \\ &\equiv m \pmod{p}, \end{aligned}$$

   o sea,

   $$m^{ed} \equiv m \pmod{p} \tag{5.3}$$

   Por otra parte, al ser $\phi(q) = q - 1$ y cumplirse $(m, q) = 1$ se tendrá en virtud Teorema B.1.3 que $m^{q-1} \equiv 1 \pmod{q}$. Al ser $\phi(n) = (p-1)(q-1)$ se cumplirá $(q-1) \mid (ed-1)$ y por tanto $m^{ed-1} \equiv 1 \pmod{q}$ de donde

   $$m^{ed} \equiv m \pmod{q} \tag{5.4}$$

   Del sistema:

   $$\begin{aligned} x &\equiv m \pmod{p} \\ x &\equiv m \pmod{q} \end{aligned}$$

   sabemos:

- Que m es solución suya.
- Por el *Teorema Chino del Resto*, que es equivalente a la ecuación:

$$x \equiv m \pmod{[p, q]}$$

- Que $[p, q] = pq = n$, puesto que $(p, q) = 1$.
- Por (5.3) y la (5.4), que m^{ed} también es una de sus soluciones.

Uniendo todo lo anterior deducimos, en este caso, que:

$$m^{ed} \equiv m \pmod{n}$$

La demostración sería la misma si m fuese múltiplo de q en lugar de p (de ambos no puede ser múltiplo debido a la condición $m < n$).

□

Ejemplo 5.2.1. Dado un texto, en primer lugar es necesario convertirlo en un número que pueda ser partido por dígitos para proceder al cifrado uno a uno. Supongamos pues que disponemos de un juego de clave RSA $K = \langle p, q, e, d \rangle$, donde $n = pq$ como se dijo. Para simplificar en el ejemplo y poder mostrar lo esencial del mismo podemos cambiar cada letra por un número con dos dígitos decimales de acuerdo con la biyección f de la Figura 1.1 de forma que si el mensaje M es dividido en bloques M_i, se cumpla $0 \leq M_i < n$. Es necesario tener $M_i < n$ porque si no, $M_i^{ed} \pmod{n}$ podría no ser igual a M_i. Si n tiene k dígitos, queda garantizado que $M_i < n$ tomando M_i con al menos $k - 1$ dígitos. Consideremos el juego de clave RSA:

$$K = \langle 1597, 1087, 187 \rangle$$

donde $n = 1597 * 1087 = 1735939$ y, como consecuencia, $\phi(1735939) = 1733256$ de donde $d = 37075$. Consideremos también la expresión:

```
repliegue al blocao a
```

a la que desproveemos de espacios en blanco:

```
repliegueal blocaoa
```

para traducirla según la biyección f de la Figura 1.1, quedando:

170415110804062004001101111402001400

y como el número de cifras de n es 7, divideremos el mensaje en bloques de 6 cifras. Así pues:

M_1	M_2	M_3	M_4	M_5	M_6
170415	110804	62004	1101	111402	1400

Cifrando tenemos:

$E_K(M_1)$	$E_K(M_2)$	$E_K(M_3)$	$E_K(M_4)$	$E_K(M_5)$	$E_K(M_6)$
1442861	0082626	0690931	1489385	0363461	0062317

Así pues, el emisor envía el mensaje cifrado:

```
1442861 0082626 0690931 1489385 0363461 0062317
```

El receptor realiza el descifrado:

- $D_K(1442861) = 170415$
- $D_K(82626) = 110804$
- $D_K(690931) = 62004$
- $D_K(1489385) = 1101$
- $D_K(363461) = 111402$
- $D_K(62317) = 1400$

y cómo el número de más dígitos obtenido es de 6, completamos todos con ceros a la izquierda hasta obtener secuencias de 6 dígitos y yuxtaponemos las secuencias de dígitos; entonces el resultado sería:

```
17041511080406200400110111114020001400
```

Esto se traduce en:

```
repliegue alblocaoa
```

que leído es:

repliegue al blocao a

Teorema 5.2.3. *Sea* n *el producto conocido de dos números primos distintos* p *y* q*. Conocer la factorización de* n *es equivalente a conocer* $\phi(n)$*.*

Demostración. Supóngase que $n = pq$, siendo p y q primos distintos. Si conocemos la factorización de n, entonces $\phi(n) = (p-1)(q-1)$. Recíprocamente, si es conocido el valor de $\phi(n)$ podemos calcular p y q. En efecto:

$$\begin{aligned}\phi(n) &= (p-1)(q-1)\\ &= (p-1)(\frac{n}{p}-1).\end{aligned}$$

Si operamos resulta:

$$p^2 + (\phi(n) - n - 1)p + n = 0$$

pero como también se cumple $\phi(n) = (\frac{n}{q}-1)(q-1)$ entonces tenemos que tanto p cómo q son soluciones de la ecuación de segundo grado:

$$x^2 + (\phi(n) - n - 1)x + n = 0$$

Así pues, determinar $\phi(n)$ es equivalente a factorizar n. □

Definición 5.2.2. Para todo número natural no nulo n, sea $\sigma(n)$ el valor defininido por la siguiente igualdad:

$$\sigma(n) = \sum_{\substack{k=1\\(k,n)=1}}^{n} k$$

Lema 5.2.4. *Para todo número natural no nulo* n*,*

$$\sigma(n) = \frac{n\phi(n)}{2}$$

Demostración. Sea k un número natural —fijo pero arbitrario— tal que $1 \leq k \leq n$ y $(k, n) = 1$; entonces existen enteros u y v tales que:

$$\begin{aligned}1 &= uk + vn\\ &= uk + 0 + vn\\ &= uk - un + un + vn\\ &= u(k-n) + (u+v)n\end{aligned}$$

de lo que deducimos que $(n-k, n) = (k-n, n) = 1$, pero para todo $1 \leq k \leq n$ se cumple que $k + (n-k) = n$ así que:

$$\sigma(n) = \frac{n\phi(n)}{2}$$

□

Corolario 5.2.5. *Sea n el producto conocido de dos números primos distintos p y q. Conocer la factorización de n es equivalente a conocer $\sigma(n)$.*

Definir los exponentes públicos y privados como inversos módulo $\phi(n)$, como originalmente se hizo en RSA, es una condición suficiente —pero no necesaria— para que el método de descifrado recupere el texto llano a partir del texto cifrado. Una condición necesaria es que los exponente públicos y privados sean inversos uno del otro módulo la *función lambda de Carmichael*, $\lambda(n)$ (cfr. Sección B.2). Esto es gracias a que la función lambda de Carmichael es, por defincón, el menor número m tal que:

$$a^m \equiv 1 \pmod{n}$$

para todo entero a que cumpla $(a, n) = 1$.

Así pues, basta con definir el exponente público y privado como inversos módulo cualquier múltiplo de $\lambda(n)$. Para un módulo RSA que sea $n = pq$, $\lambda(n) = [p, q]$. Por supuesto que $\phi(n)$ es un múltiplo de $\lambda(n)$, como queda argumentado en la Sección B.2, pero también podemos entenderlo según el siguiente argumento:

$$\begin{aligned}\phi(n) &= (p-1)(q-1) \\ &= (p-1, q-1)[p-1, q-1] \\ &= (p-1, q-1)\lambda(n)\end{aligned}$$

lo que autoriza a usar $\phi(n)$ en el algoritmo de generación del juego de claves para RSA. En determinadas implementaciones actuales de RSA, por ejemplo PKCS #1, el juego de claves es generado módulo $\lambda(n)$.

Ejercicio 5.2.1. Sean p y q, dos números primos distintos. Demuestre que si

$$ed \equiv 1 \pmod{\lambda(n)}$$

entonces d puede ser usado como un exponente de descifrado en el esquema RSA con exponente de cifrado igual a e.

5.3. Seguridad de RSA

La seguridad del sistema deriva de la creencia de que la función de cifrado $E_k(m) = m^e \pmod{n}$ es una función de una vía; esto no es un hecho probado. Se cree que determinar la función de descifrado D a partir de E es equivalente a factorizar n. Aunque esta equivalencia no ha sido probada para el sistema RSA, se ha probado que ciertas variaciones del esquema RSA son equivalentes a factorizar. Demostraremos que calcular $\Phi(n)$ ó d es equivalente a factorizar n, pero esto deja de lado el asunto de ser capaces de recuperar M de $E_K(M)$ sin conocer ni d ni $\Phi(n)$

Teorema 5.3.1. *Supongamos que* $n = pq$ *es el producto de dos primos distintos. Determinar* $\Phi(n)$ *es equivalente a factorizar* n.

Seguidamente demostraremos que si hay un método de calcular el exponente de descifrado d sin conocer $\Phi(n)$, entonces podríamos ser capaces de factorizar n, suponiendo que d satisface la relación $ed \equiv 1 \pmod{\Phi(n)}$. El algoritmo es probabilístico, en el sentido de que depende de inputs aleatorios y no está garantizado el éxito en todo momento. Demostraremos que si el procedimiento es aplicado una vez, entonces la probabilidad de éxito es al menos $1/2$. Por consiguiente, si el procedimiento es repetido k veces, entonces la probabilidad de k fallos es a lo sumo $(1/2)^k$; esto es, la probabilidad de ser capaces de factorizar el número en k intentos es al menos $1 - (1/2)^k$. En la práctica, tenemos la garantía de encontrar un factor en pocos intentos.

El algoritmo se basa en el hecho de que la ecuación $x^2 \equiv 1 \pmod{n}$ tiene cuatro soluciones cuando n es producto de dos primos distintos. Cualquier solución α de $x^2 \equiv 1 \pmod{n}$ es tal que $n \mid (\alpha - 1)(\alpha + 1)$. Siempre que $\alpha \not\equiv \pm 1 \pmod{n}$, seremos capaces de calcular un factor de n, y por tanto los dos no triviales, calculando $(\alpha - 1, n)$ ó $(\alpha + 1, n)$.

Supongamos que conocemos d tal que $ed \equiv 1 \pmod{\Phi(n)}$ y expresemos $ed - 1 = 2^r s$, donde s es impar. Ahora elijamos aleatoriamente un número tal que $0 < w < n$. Si $(w, n) \neq 1$, nuestro esfuerzo no sería necesario pues de ello tendríamos una factorización de n; supongamos por contra que $(w, n) = 1$. Entonces caculamos la secuencia:

$$z_0 = w^s$$
$$z_k = z_{k-1}^2 \bmod n,\ 1 \leq k \leq r$$

Observe que $z_k = w^{2^k s} \bmod n$. Como por hipótesis se tiene que $\Phi(n) \mid ed - 1$, en realidad esto significa que $\Phi(n) \mid 2^r s$ y, por el Teorema de Euler, que z_r es 1. Si el primer término de la sucesión no es 1, entonces habrá un número $z_k \not\equiv 1 \pmod{n}$ tal que $z_k^2 \equiv 1 \pmod{n}$. Supongamos que un tal número z_k existe cumpliendo eso y también que $z_k \not\equiv -1 \pmod{n}$, entonces habremos encontrado una raíz no trivial de la unidad módulo n y, en consecuencia, una factorización de n.

El procedimiento descrito fracasa cuando $z_0 = 1$ o cuando existe un índice k tal que $z_k = -1$. Para estimar la probabilidad de fallo, contamos el número de enteros w, $0 \leq w < n$ tal que $z_0 = 1$ o para algún k, $z_k = -1$; es decir, estamos interesados en el número de soluciones de las congruencias:

$$w^s \equiv 1 \pmod{n} \tag{5.5}$$
$$w^{2^i s} \equiv -1 \pmod{n},\ 0 \leq k < r \tag{5.6}$$

Demostraremos que el número de soluciones a estas congruencias es a lo sumo $n/2$. El hecho crucial en la demostración es el *Teorema de Lagrange*, o más bien su corolario de que el número de soluciones a la ecuación $x^d \equiv 1 \pmod{p}$ es $(p-1, d)$ cuando p es primo.

Lema 5.3.2. *Supongamos que p es un número primo y d un número entero. Sea $p - 1 = 2^r s$ y $d = 2^{r'} s'$, donde tanto s como s' son enteros impares. Entonces:*

1. *La congruencia $w^d \equiv 1 \pmod{p}$ tiene exactamente $(d, p-1)$ soluciones.*
2. *Si p es impar, la congruencia $w^d \equiv -1 \pmod{p}$ tiene exactamente $(d, p-1)$ soluciones, siempre que $r' < r$.*
3. *Si p es impar, la congruencia $w^d \equiv -1 \pmod{p}$ no tiene solución alguna, siempre que $r' \geq r$.*

Demostración. Para la **primera parte**, por el Corolario B.3.2, si p es un número primo y $d \mid (p-1)$, entonces la ecuación $x^d - 1 \equiv 0 \pmod{p}$ tiene exactamente d soluciones. Pero al ser d un divisor de $p-1$, $(d, p-1) = d$; de ahí que $x^d \equiv 1 \pmod{p}$ tenga exactamente $(d, p-1)$ soluciones. Supongamos por contra que $d \nmid (p-1)$ y sean números enteros x e y tales que

$$(d, p-1) = xd + y(p-1)$$

Por el Teorema de Fermat (cfr. Teorema B.1.1) $w^{p-1} \equiv 1 \pmod{p}$ (suponemos $w < p$); si $w^d \equiv 1 \pmod{p}$ (como suponemos) entonces $(w^d)^x (w^{p-1})^y \equiv 1 \pmod{p}$ y por tanto $w^{(d,p-1)} \equiv 1 \pmod{p}$. Por consiguiente, la congruencia $w^d \equiv 1 \pmod{p}$ es equivalente a $w^{(d,p-1)} \equiv 1 \pmod{p}$; pero ésta última, de nuevo por el Corolario B.3.2, sabemos que tiene exactamente $(d, p-1)$ soluciones. Esto demuestra que en cualaquiera de los dos casos posibles, $w^d \equiv 1 \pmod{p}$ tiene exactamente $(d, p-1)$ soluciones. Para la **segunda parte**, si $w^d \equiv -1 \pmod{p}$, entonces $w^{2d} \equiv 1 \pmod{p}$. Esta última congruencia tiene exactamente $(2d, p-1)$ soluciones. No obstante:

$$\begin{aligned}(w^d - 1)(w^d + 1) &\equiv w^{2d} - 1 \pmod{p} \\ &\equiv 0 \pmod{p}\end{aligned}$$

Ahora bien $w^d - 1 \equiv 0 \pmod{p}$ tiene exactamente $(d, p-1)$ soluciones. Entonces la ecuación $w^d + 1 \equiv 0 \pmod{p}$ tiene solución si:

$$(2d, p-1) > (d, p-1) \tag{5.7}$$

En virtud de las propiedades del máximo común divisor tenemos que:

$$(2d, p-1) = 2^{\min\{r'+1, r\}}(s', s)$$

y

$$(d, p-1) = 2^{\min\{r',r\}}(s', s)$$

La desigualdad (5.7) se traduce en la condición:

$$\min\{r', r\} < \min\{r'+1, r\}$$

Es posible sólo cuando $r' < r$. En este caso $(2d, p-1)$ es dos veces $(d, p-1)$, así que $w^d \equiv -1 \pmod{p}$ tiene exactamente $(d, p-1)$ soluciones. Si $r' \geq r$, entonces $(2d, p-1) = (d, p-1)$ y por tanto $w^d \equiv -1 \pmod{p}$ no tendrá solución alguna. □

Ejemplo 5.3.1. Sea $n = pq$, con $p = 137$ y $q = 97$. Usaremos el Lema 5.3.2 el número de soluciones de las ecuaciones (5.5) y (5.6). Tenemos que $p-1 = 2^3 \cdot 17$ y que $p-1 = 2^5 \cdot 3$. Sea $ed - 1 = 2^r s$, donde $s = 51$. La ecuación $w^s \equiv 1 \pmod{p}$ tiene $(s, p-1) = 17$ soluciones y $w^s \equiv 1 \pmod{q}$ tiene $(s, q-1) = 3$ soluciones. Por tanto, la ecuación $w^s \equiv 1 \pmod{n}$ tiene 51 soluciones. Por el Lema 5.3.2, las congruencia $w^s \equiv -1 \pmod{p}$ tiene $(s, p-1) = 17$ soluciones y $w^s \equiv -1 \pmod{q}$ tiene $(s, q-1) = 3$. Así pues, $w^s \equiv 1 \pmod{n}$ tiene 51 soluciones.

Similarmente la congruencia $w^{2s} \equiv -1 \pmod{p}$ tiene $(2s, p-1) = 34$ soluciones y $w^{2s} \equiv -1 \pmod{q}$ tiene $(s, q-1) = 6$. Así pues, $w^s \equiv 1 \pmod{n}$ tiene $34 \cdot 6 = 204$ soluciones.

La congruencia $w^{4s} \equiv -1 \pmod{p}$ tiene $(4s, p-1) = 68$ soluciones y $w^{4s} \equiv -1 \pmod{q}$ tiene $(s, q-1) = 12$. Así pues, $w^s \equiv 1 \pmod{n}$ tiene 816 soluciones.

No es posible tener $w^{8s} \equiv -1 \pmod{n}$, porque $(p-1) \mid 8s$ implica que $w^{8s} \equiv 1 \pmod{p}$. Así pues, el número total de soluciones a las congruencias (5.5) y (5.6) es $51 + 51 + 204 + 814 = 1122$, mientras que hay 13056 elementos invertibles módulo n.

Teorema 5.3.3. *Sean p y q primos impares distintos, $n = pq$ y e y d tales que $\phi(n) \mid ed - 1$. Si $ed - 1 = 2^r s$, siendo s impar, entonces el número de solucines de las congruencias:*

$$w^s \equiv 1 \pmod{n} \tag{5.8}$$

$$w^{2^i s} \equiv -1 \pmod{n},\ 0 \leq k < r \tag{5.9}$$

es a lo sumo $n/2$.

Demostración. Sean $ed - 1 = 2^r s$, $p - 1 = 2^{r_1} s_1$ y $q - 1 = 2^{r_2} s_2$, donde r, r_1 y r_2 son impares. Como $\phi(n) = (p-1)(p-2)$ divide a $ed - 1$, debe ocurrir que $2^{r_1+r_2} \mid 2^r$ y $s_1 s_2 \mid s$.

Ahora bien, $w^d \equiv 1 \pmod{n}$ si, y sólo si, $w^d \equiv 1 \pmod{p}$ y $w^d \equiv 1 \pmod{q}$. La ecuación $w^s \equiv 1 \pmod{p}$ tiene $(s, p-1) = s_1$ soluciones y $w^s \equiv 1 \pmod{q}$ tiene

$(s, q-1) = s_2$ soluciones. Por el *Teorema Chino del Resto*, el número de soluciones de $w^s \equiv 1 \pmod{pq}$ es $s_1 s_2$.

Seguidamente contamos las soluciones a $w^{2^i s} \equiv -1 \pmod{m}$. Esto es equivalente a $w^{2^i s} \equiv -1 \pmod{p}$ y $w^{2^i s} \equiv -1 \pmod{q}$. Por el *Teorema de Fermat* (cfr. Teorema B.1.1), debemos tener $i < r_1$ e $i < r_2$, es decir, $i < \min\{r_1, r_2\}$. Por el Lema 5.3.2 sabemos que la congruencia $w^{2^i s} \equiv -1 \pmod{p}$ tiene $2^i s_1$ soluciones y $w^{2^i s} \equiv -1 \pmod{q}$ tiene $2^i s_2$ soluciones; por tanto, la congruencia $w^{2^i s} \equiv -1 \pmod{pq}$ tiene $2^{2i} s_1 s_2$ soluciones.

Podemos suponer, sin pérdida de generalidad, que $r_1 < r_2$. Sumando el número de soluciones a cada una de las congruencias de la afirmación del teorema, obtenemos:

$$\begin{aligned} s_1 s_2 + \sum_{i=0}^{r_1-1} 2^{2i} s_1 s_2 &= s_1 s_2 \left(1 + \frac{4^{r_1} - 1}{4 - 1}\right) \\ &= s_1 s_2 \frac{4^{r_1} + 2}{3} \end{aligned}$$

Es simple demostrar que esta suma es menor que $\phi(n)/2$, de donde es menor que $n/2$. □

5.4. Ataques y Defensa del Protocolo RSA

Se han propuesto diversos ataques frente al sistema RSA. Sin embargo, ninguno de estos ataques ha sido considerado una amenza seria que permita desaconsejar su uso. De todas formas debe tenerse en cuenta que el aumento en la potencia de computación es una amenza para todo sistema, si bien en el caso de RSA también puede servir para fortificarlo con lo que, en cierto sentido, la amenaza quedaría neutralizada. Son más de temer —o de desear— los avances algorítmicos en aspectos como la factorización de números enteros.

En lo que a n respecta, hay que tener en cuenta que nuestra criptosistema concreto *RSA* está al descubierto en cuanto que el número n —que es público— sea factorizado. Esta es la modalidad de ataque frente a *RSA* más popularizada y *Bruce Schneier* afirma en [17] que "la seguridad de *RSA* depende enteramente del problema de factorización de números elevados". En líneas posteriores, este autor matiza: "técnicamente esto no es cierto. Se ha conjeturado que la seguridad del *RSA* depende del problema de factorización de números elevados. Nunca ha sido demostrado matemáticamente que necesitas factorizar n para calcular m a partir de c y e.[2] Es concebible que pudiera ser

[2]Se refiere el autor con m al texto llano, con c al texto cifrado y con e a la segunda componente de la llave pública

descubierta una forma de criptoanálisis del *RSA* enteramente diferente". Lo que sí es de destacar es que esta posible nueva forma de criptoanálisis proporcionaría una nueva forma de factorización.

De los Tamaños Relativos de los Factores

Un número de 120 dígitos es por lo general factorizable y los de 200 dígitos se prevé que serán factorizables pronto. Por tanto, n debe superar en longitud a 120. Algunas implementaciones del *RSA* emplean números de 154, 200 o incluso 308 dígitos.

La primera precaución que debe tomarse al elegir n es que sus factores sean lo más elevados posible. Esto implica que debe tener un número mínimo de factores sin ser primo, o sea, debe ser producto de dos primos. Los primos deben ser elegidos "aleatoriamente" y en ningún caso pertenecer a una tabla conocida, dado que ésta podría ser de mucha utilidad en la factorización. En cualquier caso, y de cara al ataque, se puede intentar la factorización de n probando con primos de una forma conocida o recogidos en algunas tablas.

Los dos factores primos de n, p y q, no deben ser próximos uno al otro. En efecto, si lo fueran —supongamos para fijar ideas que $p > q$— entonces $(p-q)/2$ sería pequeño y $(p+q)/2$ sería ligeramente superior a $\sqrt{n}$. Por otra parte,

$$\frac{(p+q)^2-(p-q)^2}{4} = n$$

de donde $\frac{(p+q)^2}{4} - n$ es el cuadrado de un entero. Para factorizar n tomaríamos enteros $x > \sqrt{n}$ hasta encontrar uno tal que exista otro y verificando

$$x^2 - n = y^2$$

En tal caso $p = x + y$ y $q = x - y$.

En el párrafo anterior lo que se da en realidad es un método sencillo de factorizar enteros, el conocido como método de *factorización de Fermat*, junto a la observación de una circunstancia que hace sencilla la factorización de n por este método. Considérese, por ejemplo, el caso de $n = 97343$. Se tiene que $\lfloor\sqrt{n}\rfloor = 311$ y que $312^2 - n = 1$, lo que da directamente que $p = 313$ y $q = 311$. No obstante, los factores primos de n no debe de estar muy separados en tamaño. De ser uno muy pequeño, sería fácilmente encontrable por la aplicación de la *criba de Eratóstenes*. Es aconsejable que los factores de n, representados en binario, difieran en unos pocos bits.

Fallo del Protocolo por el Módulo Común

Supongamos que en un círculo RSA dos usuarios, Alice y Bob, se atribuyen respectivamente las claves públicas $\langle n, e_1\rangle$ y $\langle n, e_2\rangle$ en un círculo de usarios de RSA de forma que $(e_1, e_2) = 1$. Supongamos que un tercer miembro del círculo, digamos Carol, cifra un mismo mensaje m para Alice y Bob. Entonces m puede ser leído sin conocer la clave de descifrado ni la factorización de n. Supogamos que ha sido difundido:

- m^{e_1} mód n
- m^{e_2} mód n

Si $(e_1, e_2) = 1$, existen x e y tales que $e_1x + e_2y = 1$; entonces:

$$\begin{aligned} m &= m^1 \\ &= m^{e_1x+e_2y} \\ &= (m^{e_1})^x(m^{e_2})^y \text{ mód } n \end{aligned}$$

Ejemplo 5.4.1. Por ejemplo, supongamos que $n = 5038301$ es el módulo común entre varios usuarios. Sean $e_1 = 787$ y $e_2 = 6785$. El algoritmo extendido de Euclides proporciona la igualdad $888e_1 - 103e_2 = 1$. Si tenemos:

- m^{e_1} mód $n = 986536$ y
- m^{e_2} mód $n = 2294886$

entonces $m = (986536^{888})(2294886^{-103})$ mód $n = 52013$. De 052013 recuperamos entonces la palabra:

`fun`

por la correspondencia acordada entre letras y números. Considérese el siguiente diálogo de `sagemath`:

```
sage: n, e1, e2 = 5038301, 787, 6785
sage: xgcd (e1,e2)
(1, 888, -103)
sage: power_mod(986536,888,n) * power_mod(2294886,-103,n) % n
52013
sage: (986536**888)*(2294886)**(-103) % n
52013
```

Fallo del Protocolo por Elección de Exponentes Bajos

Si d es muy pequeño entonces sería fácil de encontrar por el procedimiento de prueba y error. Esta es realmente la razón de comenzar elegiendo d y luego proceder a determinar e. De esa forma podremos seleccionar un valor d razonablemente elevado. Sin embargo, no es nada conveniente tener valores pequeños de e, como demuestra *M.J. Wiener* en su trabajo [19].

Aparte de los resultados aportados por *Wiener*, imaginemos el caso en que un individuo del círculo opera con un exponente e ($e \geq 2$) de cifrado excesivamente bajo, verbigracia 3, y que un número superior o igual a e de compañeros han hecho la misma elección, dejándose llevar quizá por la facilidad que supone cifrar con e. Eso sí, cada uno de ellos se ha preocupado de elegir un módulo de cifrado que es primo relativo con cualquiera de los módulos que conoce; de no ser así su módulo sería fácilmente factorizado y la secrecidad destruida. Si un oponente detecta los valores $c_1, \ldots, c_e$ que corresponden al cifrado de un mismo mensaje m enviado a e de los miembros que comparten e como exponente de cifrado, entonces puede considerar el siguiente sistema de congruencias:

$$\begin{aligned} x &\equiv c_1 \pmod{n_1} \\ x &\equiv c_2 \pmod{n_2} \\ &\vdots \\ x &\equiv c_e \pmod{n_e} \end{aligned} \tag{5.10}$$

y como $(n_i, n_j) = 1$, para todo $1 \leq i < j \leq e$, resulta por el *Teorema Chino del Resto* que el sistema (5.10) tiene una única solución w tal que $0 \leq w < \prod_{i=1}^{e} n_i$. Como para todo $1 \leq i \leq e$ se cumple $m < n_i$, resulta entonces que $m^e < \prod_{i=1}^{e} n_i$ y dado que m^e es solución de (5.10), entonces $w = m^e$. De este forma el oponente puede leer el mensaje resolviendo el sistema por el algoritmo adecuado, lo cual no es excesivamente costoso, y extrayendo la raíz e-ésima a la solución obtenida.

Ejemplo 5.4.2. Consideremos un círculo de usuarios RSA que han publicado las siguientes señas $\langle n, e \rangle$:

- $K_0 = \langle 6689477, 3 \rangle$
- $K_1 = \langle 5346437, 3 \rangle$
- $K_2 = \langle 7675039, 3 \rangle$

El mensaje es m ha sido cifrado para cada uno de los usuarios, resultando que: $y_0 = 5886824$, $y_1 = 2815816$ e $y_2 = 418668$. Consideramos entonces el sistema:

$$\begin{aligned} x &\equiv 5886824 \pmod{6689477} \\ x &\equiv 2815816 \pmod{5346437} \\ x &\equiv 418668 \pmod{7675039} \end{aligned}$$

cuya solución es:

$$x \equiv 140713482366197 \pmod{274496751690797469511}$$

y extrayendo la raíz cúbica a 140713482366197 resulta 52013 que debe ser entendido como:

```
fun
```

y que efectivamente es el mensaje original cifrado y enviado a los tres usuarios, como se puede comprobar.

Del Valor de $\phi(n)$

Es claro que $p-1$ y $q-1$ no son primos relativos, de hecho su máximo común divisor es un múltiplo de 2. Para d existe $t \in \mathbb{N}$ tal que $de = 1 + t(p-1)(q-1)$, o sea, $de = 1 + t(p-1, q-1)[p-1, q-1]$. De ello se deduce que d es un inverso de e módulo $[p-1, q-1]$. Un posible ataque sería proponerse distintos valores m candidatos a $[p-1, q-1]$ de forma que e tenga inverso d' módulo m, seguidamente generaríamos una secuencia de valores del tipo $d_t = d' + tm$ $(t \in \mathbb{N}^*)$ y probaríamos a descifrar con d_t.

Si $(p-1, q-1)$ es muy elevado entonces $[p-1, q-1]$ será muy pequeño en comparación con $\phi(n)$ y alcanzar su valor por prueba y error será más fácil. En definitiva, encontraremos alguna facilidad en encotrar d generando los valores d''. El caso extremo se da cuando $p-1 \mid q-1$, entonces basta buscar inversos módulo $q-1$ probando con distintos candidatos a $q-1$.

Ejemplo 5.4.3. Este es el caso de $n = 11041 = 61 \cdot 181$. Puesto que $60 \mid 180$, basta buscar d entre los inversos de e módulo 180. Supongamos que $e = 4013$. Como

$$1 = 10800 \cdot 1532 - 4013 \cdot 4123$$

tendremos que $d = 6677$. Por otra parte, probando con valores posible de $[p-1, q-1]$ llegaríamos pronto a 180 y entonces calcularíamos $d' = e^{-1} \bmod 180$. Dado que

$$1 = 17 \cdot 4013 - 180 \cdot 379$$

se tendrá que $d' = 17$ y $d'_{37} = d$.

Otra característica indeseable en n es que los factores primos de $\phi(n)$ sean pequeños, por ejemplo todos menores que un natural k pequeño. Si $r \leq k$ es un primo tal que $r \mid \phi(n)$, el valor máximo de i tal que $r^i \mid \phi(n)$ es un número menor que $\lfloor \log_r n \rfloor$. En esta ocasión podría tener éxito un ataque que consistiera en construir un amplio número de candidatos v a ser $\phi(n)$. En cada caso se intentaría el criptoanálisis "elevando el texto cifrado" a $(v+1)/e$, cuando este fuera un valor entero.

Para solventar las debilidades que una mala elección de n en los dos sentidos anteriores pudiera infundir el criptosistema, deberíamos asegurarnos de que los primos p y q elegidos fueran *primos seguros*.

Definición 5.4.1. p es un *primo seguro* si cumple:

1. p es primo.
2. $\frac{p-1}{2}$ es primo.

Como ejemplos de primos seguros tenemos 83, 107 y $10^{100} - 166517$. Es evidente que la generación de primos seguros es más difícil que la generación de primos, además no se sabe si existen infinitos o no.

De los Cifrados Iterados

Otra modalidad de ataque es el conocido como el de *cifrados iterados*. Se basa en tomar un texto cifrado c y construir los distintos términos de la sucesión $\{a_i\}_{i\in\mathbb{N}}$ definida como sigue:

$$a_i = \begin{cases} c, & \text{si } i = 0 \\ a_{i-1}^e \bmod n, & \text{si } i > 0 \end{cases}$$

Nos detenemos cuando para $0 < j$ se cumpla $c_j = c_0$. Entonces el mensaje que busca el oponente es c_{j-1}.

Rivest ha demostrado (cfr. [14]) que si los enteros $p-1$ y $q-1$ tienen factores primos elevados —por ejemplo, $q-1 = 2q'$ y $p-1 = 2p'$ siendo p' y q' primos— entonces la probabilidad de éxito en el ataque es cercana a 0 cuando $n = pq$ es elevado. Por ejemplo, si p' y q' son mayores que 10^{90}, entonces la probabilidad de éxito en un ataque de cifrados iterados es cercana a 10^{-90}.

Ejemplo 5.4.4. Supongamos que A cifra con RSA y tiene la llave pública $\langle 35, 17\rangle$. Supongamos que ha recibido el mensaje $3 = p^{17} \text{ mód } 35$ y que lo hemos interceptado. Componemos la sucesión y encontramos que

$$\begin{aligned} c_0 &= 3 \\ c_1 &= 3^{17} \text{ mód } 35 = 33 \\ c_2 &= 33^{17} \text{ mód } 35 = 3 \end{aligned}$$

El mensaje p será 33.

De los Mensajes Inocultables

Blakley y *Borosh* pusieron de manifiesto en [4] que los sistema *RSA* no siempre ocultan cualquier mensaje. Pongamos el caso de un sistema criptográfico *RSA* con la llave pública $\langle 35, 17\rangle$ mediante el cual se desea ocultar cualquier mensaje compuesto con los números del conjunto:

$$I = \{1, 6, 7, 8, 13, 14, 15, 20, 21, 22, 27, 28, 29, 34\}$$

Encontraremos que por desgracia para todo $x \in I$, $x = x^{17}$ mód 35. Más dramática aún es la situación en la que $p = 97$, $q = 109$ y $e = 865$. En este caso es imposible esconder mensaje alguno.

Lema 5.4.1. *El número de raíces n-ésimas de la unidad en* $GF(q)$ *es* $(n, q-1)$.

Teorema 5.4.2. *Sea el criptosistema RSA con llave pública* $\langle n, e\rangle$, *donde* $n = pq$. *Existen*

$$\sigma_e = \big(1 + (e-1, p-1)\big)\big(1 + (e-1, q-1)\big)$$

elementos $0 \leq m < n$ *tales que* $m^e \equiv m \pmod{n}$.

Demostración. Sea $0 \leq m < n$. El que se cumpla $m^e \equiv m \pmod{pq}$ es equivalente a que se cumplan simultaneamente las relaciones:

$$\begin{aligned} m^e &\equiv m \pmod{p} \\ m^e &\equiv m \pmod{q} \end{aligned} \tag{5.11}$$

La ecuación $m^e \equiv m \pmod{p}$ puede ser reescrita como $m^{e-1} \equiv 1 \pmod{p}$ en el caso de que $(m, p) = 1$ y como $m^{e-1} \equiv 0 \pmod{p}$ en otro caso —lo mismo se puede decir respecto a $m^e \equiv m \pmod{q}$—. Por el lema 5.4.1 sabemos que la $m^{e-1} \equiv 1 \pmod{p}$ tiene $(e-1, p-1)$ soluciones, de donde el sistema 5.11 tendrá $\big(1+(e-1, p-1)\big)\big(1+(e-1, q-1)\big)$ soluciones.

□

Ejemplo 5.4.5. Sean $p = 5$ y $q = 7$, con lo cual $n = 35$. Consideremos los siguientes casos:

1. Si $e = 2$, entonces $\sigma_e = 4$ y los mensajes inocultables son $\{0, 1, 15, 21\}$.
2. Si $e = 3$, entonces $\sigma_e = 9$ y el conjunto de mensajes inocultables es
$$\{0, 1, 6, 14, 15, 20, 21, 29, 34\}$$
3. Si $e = 5$, entonces $\sigma_e = 15$ y el conjunto de mensajes inocultables es
$$\{0, 1, 6, 7, 8, 13, 14, 15, 20, 21, 22, 27, 28, 29, 34\}$$
4. Si $e = 7$, entonces $\sigma_e = 21$ y el conjunto de mensajes inocultables es
$$\{0, 1, 4, 5, 6, 9, 10, 11, 14, 15, 16, 19, 20, 21, 24, 25, 26, 29, 30, 31, 34\}$$

Observación 5.4.1. Obsérvese que si $e \equiv 3 \bmod \phi(n)$ entonces existe t tal que $e - 1 = (p - 1)(q - 1)t + 2$, entonces $(e - 1, p - 1) = (2, p - 1) = 2$ y análogamente $(e - 1, q - 1) = (2, q - 1) = 2$.

Corolario 5.4.3. *Sea el criptosistema RSA de la llave pública* $\langle n, e\rangle$*, donde* $n = pq$*. Entonces*

1. *Si* $(e - 1, p - 1) = (e - 1, q - 1) = 1$*, entonces* $\sigma_e = 4$*.*
2. *Si* $(e - 1, p - 1) = (e - 1, q - 1) = 2$*, entonces* $\sigma_e = 9$*.*
3. *Si* $e \equiv 3 \bmod \phi(n)$*, entonces* $\sigma_e = 9$*, con independencia de los valores de* p *y* q*.*
4. *Si* $[p-1, q-1] \mid e-1$*, entonces* $\sigma_e = n$*. Así pues, la elección de* $e = \phi(n)/2+1$*, si ha lugar a ser hecha, es especialmente mala.*

Observación 5.4.2. Si hacemos que tanto p como q sean primos perfectos de la forma $2p' + 1$ y $2q' + 1$ respectivamente, entonces $(e - 1, p - 1) = (e - 1, 2p')$ puede tomar tres valores: 1, 2 y p'.

1. Si es 1, $\sigma_e = 4$.
2. Si es 2, $\sigma_e = 9$.
3. Si es p', $\sigma_e = 2(p' + 1)$. La probabilidad de que ésto ocurra es aproximadamente $1/p$.

Resumen de Precauciones

Por tanto, a la hora de confeccionar un sistema de cifrado RSA para un determinado círculo de usuarios, conviene adoptar algunas precauciones si se pretende que el sistema sea difícil de atacar. Entre ellas están las siguientes:

1. El número n debe superar los 308 dígitos (a comienzos del siglo 21).
2. El número n debe tener exactamente dos factores primos y éstos deben ser elevados.
3. Los factores primos de n jamás deben ser elegidos de entre una lista de primos famosos.
4. Los factores primos de n no deben ser próximos el uno al otro.
5. Si los factores primos de n son p y q, $(p-1, q-1)$ no debe ser elevado en exceso y nunca debe ocurrir que $p-1 \mid q-1$.
6. Es indeseable en n que los factores primos de $\phi(n)$ sean demasiado pequeños.
7. Sería deseable que los dos factores primos de n fuesen *primos seguros*. (cfr. definición posterior)
8. n deber ser elevado y tanto $p-1$ como $q-1$ deben tener factores primos elevados. Por ejemplo que $p-1$, que tiene como factor al 2 tenga un único factor primo distinto de 2 y que éste sea elevado (por ejemplo mayor que 10^{90}, e igual para $q-1$.
9. d debe ser elevado y ésta es la razón de comenzar eligiéndolo para luego determinar e.
10. No es recomendable tener valores pequeños de e y, en caso de tenerlo, jamás debe ser elegido por varios compañeros. El *Teorema Chino del Resto* sería letal para la seguridad de nuestro círculo.
11. Cuidarse de los mensajes inocultables.

5.5. Llave Pública e Intercambio de Llaves.

El sistema de cifrado RSA se basan en el hecho de que encontrar dos primos elevados y multiplicarlos para formar un entero muy elevado n supone un esfuerzo notablemente

menor que el contrario, esto es, dado un entero muy elevado n, factorizarlo. Existe otro proceso con esta propiedad, que podríamos denominar "*ser en una dirección*", relacionado ahora con las potencias en cuerpos finitos.

Cuando tratamos con números reales, la exponenciación no es significativamente más sencilla de efectuar que su operación inversa, el logaritmo. Pero supongamos que tratamos en lugar de $\mathbb{R}$ con un grupo finito, como puede ser $GF(q)^*$ o el grupo de las unidades de $\mathbb{Z}_n$ (cuyas operaciones de grupo son en los dos casos la multiplicación). Con el metódo de exponenciación rápida calculamos de forma relativamente rápida b^x, aunque x sea elevado. Pero, si se nos da un elemento y, que sabemos que es de la forma b^x para cierto x, ¿cómo podemos calcular x?, es decir, ¿cómo podemos calcular $\log_b y$ (aquí log tiene un significado distinto pero parecido al caso de $\mathbb{R}$)?. Esta cuestión se denomina "*el Problema del Logaritmo Discreto*". La palabra "discreto" es un distintivo del ambiente en que se plantea el problema, a saber, un grupo finito, respecto al ambiente continuo que supone $\mathbb{R}$.

Definición 5.5.1. Sea $\mathcal{G}$ un grupo finito, $b \in G$ e $y \in G$ tal que y es una cierta potencia de b. El *logaritmo discreto*, de y en la base b, notado como $\log_b(y)$, es cualquier entero x tal que $b^x = y$.

Ejemplo 5.5.1. Si tomamos $\mathcal{G} = GF(19)^*$ y $b = 2$ entonces

$$\log_2(7) = 6$$

Ejemplo 5.5.2. En $GF(9)^*$, si tomamos α una raíz de $x^2 - x - 1$ entonces

$$\log_\alpha(-1) = 4$$

Observación 5.5.1. Si se desea encontrar algoritmos para resolver este problema, puede consultarse la obra [8].

Sistema de Diffie-Hellman.

El primer sistema de cifrado de llave pública inventado es el de Diffie-Hellman. Se basa en la dificultad para calcular el logaritmo discreto en el caso de cuerpos finitos de elevado cardinal. Este algoritmo puede se usado para intercambio de llaves, pero no para el cifrado de mensajes.

El método de *Diffie-Hellman* procede de la siguiente forma.

1. Los usuarios A y B eligen conjuntamente dos enteros elevados, n y g, tales que $1 < g < n$. La pareja $\langle n, g \rangle$ no tiene que por qué ser secreta y puede ser compartida por más usuarios.

2. El usuario A elige aleatoriamente un número elevado x y calcula $X = g^x$ mód n.

3. El usuario B elige aleatoriamente un número elevado y y calcula $Y = g^y$ mód n.

4. Los usarios A y B se intercambian los valores X e Y manteniendo secretos los exponentes de los que se sirvieron para generarlos.

5. El usuario A calcula $k = Y^x$ mód n.

6. El usuario B calcula $k' = X^y$ mód n.

k es la llave secreta que calculan independientemente y que comparten los usuarios A y B. Este esquema puede ser extendido a más de dos usuarios.

Observación 5.5.2. Obsérvese que $k = k' = g^{xy}$ mód n y que aunque sean conocidos: n, g, X e Y, no será posible encontrar los valores x o y a menos que se sepa calcular el logaritmo discreto —lo cual no es generalmente fácil—. La llave k se utilizará ahora de la forma en que se parezca:

1. Expresándolo en base 2 y tomando la cadena de bits que se considere necesaria para cifrar con DES.

2. Tomarlo como llave de un sistema de cifrado clásico.

Es aconsejable que la elección de g y n esté sujeta a las siguientes condiciones, si se quiere alcanzar un adecuado grado de seguridad:

1. n debe ser un primo seguro.

2. El tamaño de n debe ser de 512 bits o incluso 1028.

3. g es un elemento primitivo del cuerpo $GF(n)$, o sea, un generador del grupo cíclico $GF(n)^*$.

Ejemplo 5.5.3. Se toma $n = 83$ y $g = 2$. 83 es un primo seguro que puede ser expresado como $2 \cdot 41 + 1$. Por otra parte 2 es un elemento primitivo de $GF(83)$, dado que $2^2 = 4$ mód 83 y $2^{41} \equiv 82$ mód 83. Ambos valores son distintos de 1 en $GF(83)$. Tenemos la llave pública $\langle 83, 2\rangle$. Jordi, el usuario A, escoge $x = 59$ y Gundelina, la usuaria B, escoge $y = 41$. De esta forma Esmeregildo recibe el número 82 y Gundelina recibe 53. Ahora cada uno elevará el valor recibido a su propio número secreto y así obtener la llave compartida que no es otra que 82.

El Criptosistema de Massey-Omura.

Este ingenioso sistema procede como sigue:

1. Comenzamos eligiendo un cuerpo finito de cardinal primo muy elevado $GF(q)$.
2. Cada usuario elige aleatoriamente un número secreto $0 < e < q-1$ tal que $(e, q-1) = 1$.
3. Cada usurario calcula $d = e^{-1}$ mód $q-1$ para su correspondiente valor de e.
4. Si el usuario A quiere enviar el mensaje m al usuario B entonces le envia m^{e_A} mód q, lo cual no tiene ningún significado para B.
5. B devuelve a A el valor $s = m^{e_A e_B}$ mód q, quien compone el valor $r = s^{d_A}$ mód q.
6. B compone el mensaje r^{d_B}, lo que le proporciona m.

Observación 5.5.3. Es claro que este sistema requiere que B devuelva el mensaje cifrado a A acompañado de una firma. Es claro también que la seguridad de este sistema reside en la dureza del problema del logaritmo discreto.

El Criptosistema de ElGamal.

Este ingenioso sistema genera la llave como sigue:

1. Comenzamos eligiendo un cuerpo finito de cardinal muy elevado $GF(p)$ (con p primo) y un elemento $g \in GF(p)^*$. Es preferible, aunque no absolutamente necesario, que g sea un elemento primitivo. Suponemos que las unidades de texto llano estarán expresadas en número pertenecientes a $GF(p)$.
2. Cada usuario elige a aleatoriamente y tal que $0 < a < p-1$.
3. La llave de descifrado será a y la pública de cifrado será $y = g^a$ mód p.

El cifrado del mensaje m que se quiere enviar a A se haría como sigue:

1. Se elige aleatoriamente un entero k tal que $(k, p-1) = 1$.
2. Se envia a A la pareja $\langle g^k \text{ mód } p, mg^{ak} \text{ mód } p\rangle$. Obsérvese que g^{ak} se puede componer sin conocer a, basta y sobra con conocer el valor público g^a.

El descifrado de $\langle g^k \bmod p, mg^{ak} \bmod p\rangle$ lo realizaría A como sigue:

1. Elevando g^k a la potencia $p-(a+1)$ y obteniendo el valor r.
2. Multiplicando mg^{ak}, la segunda componente del par que recibe, por r.

Observación 5.5.4. Es aclarativo el siguiente simil. El mensaje que se envía a A consta del texto llano enmascarado y de una herramienta para quitar la máscara, herramienta que sólo puede ser usada por quién conozca a. El que sepa resolver el problema del logaritmo discreto puede encontrar a del valor g^a. Se ha congeturado que la fortaleza de este sistema reside en la dificultad de resolver el problema del logaritmo discreto.

Firma Digital Estandar (DSS).

En 1991, el NIST del gobierno americano propuso un esquema estandar de firma digital. La base que utilizó el NIST para su esquema es el logaritmo discreto y cuenta con el concepto de *función hash*. Una *función hash* es una función f que transforma un input muy elevado —de 10^6 bits— en una lista de 150 ó 200 bits. Además sobre la función hash se tiene la hipótesis de que es computacionalmente muy difícil encontrar x y x' distintos tales que $f(x) = f(x')$.

Teorema 5.5.1. *Sean p y q dos primos tales que $p \equiv 1 \pmod q$. Si $g \in \mathbb{Z}_p^*$ cumple $g^{\frac{p-1}{q}} \not\equiv 1 \pmod p$ entonces el orden del elemento $g^{\frac{p-1}{q}}$ en el grupo $\mathbb{Z}_p^*$ es q.*

Demostración. Sea $0 < g \leq p-1$ y sea $\alpha = g^{\frac{p-1}{q}}$. Como $\alpha^q = g^{p-1}$ y el grupo $\mathbb{Z}_p^*$ tiene $p-1$ elementos, se tiene que $\alpha^q \equiv 1 \pmod p$. De esto se deduce que el orden de α en $\mathbb{Z}_p^*$, $o(\alpha)$, divide a q. Pero, q es un primo por tanto $o(\alpha) = 1$ o $o(\alpha) = q$. La primera posibilidad hay que descartarla porque hemos supuesto que $g^{\frac{p-1}{q}} \not\equiv 1 \pmod p$. Así pues, $o(\alpha) = q$. □

Corolario 5.5.2. *Sean p y q dos primos tales que $p \equiv 1 \pmod q$. Si $g \in \mathbb{Z}_p^*$ cumple $g^{\frac{p-1}{q}} \not\equiv 1 \pmod p$ entonces $g^{\frac{p-1}{q}}$ es un generador del único subgrupo de $\mathbb{Z}_p^*$ con q elementos.*

Demostración. El grupo $\mathbb{Z}_p^*$ es cíclico. Por tanto, todo subgrupo suyo es cíclico y como dos grupos cíclicos con el mismo orden son isomorfos, entonces existe un único subgrupo de $\mathbb{Z}_p^*$ con orden q. Por el teorema 5.5.1, si g cumple las condiciones del enunciado entonces $g^{\frac{p-1}{q}}$ es un generador del mencionado subgrupo de $\mathbb{Z}_p^*$. □

En el uso del algoritmo de firma estandard (DSA) podemos distinguir tres fases:

1. Generación de las llaves:
   a) Generación de la llave pública.
   b) Generación de la llave privada.
2. Firma.
3. Verificación de la firma.

El usuario A generaría las llaves de la siguiente forma:

1. Elige un primo q de aproximadamente 160 bits. Para ello utiliza un generador aleatorio de números y uno o varios tests de primalidad.
2. Se elige un nuevo primo p tal que $p \equiv 1 \pmod{q}$ con aproximadamente 512 bits.
3. Se elige g_0 no nulo y menor que p tal que $g_0^{\frac{p-1}{q}} \not\equiv 1 \bmod p$. Llamaremos g al valor $g_0^{\frac{p-1}{q}} \bmod p$.
4. Se elige un entero aleatoriamente cumpliendo $0 < x < q$, que será la *llave secreta*.
5. La *llave pública* será $y = g^x \bmod p$.

Si el usuario A pretende firmar un documento entonces procede como sigue:

1. Aplica una función hash al texto llano obteniendo el valor h que cumple $0 < h < q$.
2. Se genera aleatoriamente un entero k tal que $0 < k < q$.
3. Se calcula $r = (g^k \bmod p) \bmod q$.
4. Se calcula $k^{-1} \bmod q$.
5. Se calcula $s = \left(k^{-1}(h + xr)\right) \bmod q$.

La firma sería $\langle r, s \rangle$.

Si se el usuario B quiere verificar la firma, debe proceder como sigue:

1. Se calcula $w = s^{-1} \bmod q$.
2. Se calcula $u_1 = hw \bmod q$.

3. Se calcula $u_2 = rw \bmod q$.

4. Se calcula $v = \left((g^{u_1} y^{u_2}) \bmod p\right) \bmod q$.

La firma queda verificada solamente en el caso de que $v = r$.

Teorema 5.5.3. *Con los valores de v y r de DSS, se cumple $v = r$.*

Demostración. En efecto,

$$\begin{aligned} g^{u_1} y^{u_2} \bmod p =& g^{u_1} g^{x u_2} \bmod p \\ =& \left(g^k\right)^{h(h+xr)^{-1} \bmod q} \left(g^k\right)^{xr(h+xr)^{-1} \bmod q} \bmod p \\ =& \left(g^k\right)^{(h+xr)(h+xr)^{-1} \bmod q} \bmod p \\ =& \left(g^k\right)^{1+tq} \bmod p \\ =& (g^k \bmod p)\left(g^q \bmod p\right)^{tk} \end{aligned} \tag{5.12}$$

Ahora bien, $g^q = g_0^{p-1}$ y $(g_0, p) = 1$. El *teorema pequeño de Fermat* asegura que $g^q \equiv 1 \pmod p$. De 5.12 se deduce que

$$g^{u_1} y^{u_2} \equiv g^k \bmod p \tag{5.13}$$

De ésto se deduce inmediatamente el resultado. □

Para acelerar el funcionamiento de la implementación de este sistema cabe generar aleatoriamente una cadena de valores k y otra con sus correspondientes inversos.

DSA y ElGamal.

Supongamos que el algoritmo DSA está implementado como una función que podríamos representar como $\mathrm{DSAsign}(p, q, g, k, x, h; r, s)$ y que a la que basta aportar los datos anteriores al punto y coma para que devuelva los valores r y s.

Si con los parámetros del Sistema de ElGamal invocamos la función DSAsign de la siguientes forma

$$\mathrm{DSAsign}(p, p, g, a, 0, 0; r, s)$$

resulta que r es la primera componente del par codificado a enviar. Para obtener la segunda componente volvemos aplicar el algoritmo DSA esta vez como sigue:

$$\mathrm{DSAsign}(p, p, y, k, 0, 0; r, s)$$

y tomamos de nuevo el valor r, que renombramos como u. Ahora invocaríamos:

$$\mathrm{DSAsign}(p, p, m, 1, u, 0; r, s)$$

El valor s es la segunda componente del par que se envía en el criptosistema de ElGamal.

El descifrado es igualmente sencillo. Supongamos que recibimos el par $\langle \alpha, \beta \rangle$. Tomamos la llave secreta de descifrado a e invocamos la función DSA como sigue:

$$\mathrm{DSAsign}(p, p, \alpha, a, 0, 0; r, s)$$

El valor r es $g^{k(q-a-1)}$ mód p. Si lo renombramos como e y volvemos a invocar DSA como:

$$\mathrm{DSAsign}(p, p, 1, e, \beta, 0; r, s)$$

Entonces el valor s es el texto llano m.

El método de cifrado RSA también se puede simular con DSA.

5.6. Algoritmos de Mochila.

Estos algoritmos fueron diseñados, como un modo de cifrado con llave pública, por *Ralph Merkle* y *Martin Hellman*. Aunque en principio no se utilizaban más que para cifrar, *Shamir* los adaptó para firmar. Posteriormente se encontró que el *Algoritmo de Mochila* es inseguro, sin embargo es un buen ejemplo de como aplicar un problema NP-completo a la criptografía de llave pública.

Definición 5.6.1. Un *problema de la mochila* es una terna:

$$\langle k, \langle v_0, \ldots, v_{k-1} \rangle, V \rangle \tag{5.14}$$

donde:

1. $k \in \mathbb{N}^*$,
2. $v_i \in \mathbb{N}^*$, para todo $0 \leq i \leq k-1$,
3. $V \in \mathbb{N}$.

Solucionar el problema 5.14 es encontrar, si existe, un entero

$$n = (\epsilon_{k-1}\epsilon_{k-2}\cdots\epsilon_1\epsilon_0)_2$$

(donde $\epsilon_i \in \{0, 1\}$, para todo $0 \leq i \leq k-1$) tal que

$$V = \sum_{i=0}^{k-1} \epsilon_i v_i$$

input: $k \in \mathbb{N}^*$, $\langle v_0, \ldots, v_{k-1} \rangle \in (\mathbb{N}^*)^k$ supercreciente y $V \in \mathbb{N}^*$.
output: Una sol., n, si existe, al problema supercreciente de mochila.
procedure SKP(k,$\langle v_0, \ldots, v_{k-1} \rangle$,$V$;$n$)
begin
01.) $W \leftarrow V$
02.) $j \leftarrow 1$
03.) **while** $W > 0$ **and** $j \leq k$ **do**
04.) **if** $v_{k-j} > W$ **then** $\epsilon_{k-j} \leftarrow 0$
05.) **else do**
06.) $\epsilon_{k-j} \leftarrow 1$
07.) $W \leftarrow W - v_{k-j}$
od
08.) $j \leftarrow j + 1$
od
09.) **if** $W > 0$ **then return** "no hay sol."
10.) **if** $W = 0$ **and** $j \leq k$ **then do**
11.) **for** $i = j$ **to** k **do**
12.) $\epsilon_{k-i} \leftarrow 0$
od
13.) **return** $(\epsilon_{k-1}\epsilon_{k-2}\cdots\epsilon_0)_2$
od
14.) **return** $(\epsilon_{k-1}\epsilon_{k-2}\cdots\epsilon_0)_2$
end

Figura 5.1: Algoritmo SKP para solucionar, si es posible, un problema supercreciente de mochila.

Definición 5.6.2. Sea $k \in \mathbb{N}^*$ y $\langle v_0, v_1, \ldots, v_{k-1} \rangle$ una sucesión finita tal que $v_i \in \mathbb{N}^*$, para todo $0 \leq i \leq k-1$. Dicha sucesión es supercreciente si se cumple para todo $1 \leq i \leq k-1$ la relación

$$\sum_{j<i} v_j < v_i$$

Un *problema de mochila supercreciente* en un problema de mochila tal que la sucesión $\langle v_0, \ldots, v_{k-1} \rangle$ es supercreciente.

El problema general de la mochila es equivalente al problema del *vendedor ambulante* y por tanto es un problema NP-completo. Se ha conjeturado que no existe un algoritmo que resuelva un problema arbitrario de mochila en tiempo polinomial en k y en $\log B$, donde B es una cota de V y los v_i.

No obstante, el problema de mochila supercreciente es mucho más fácil de resolver. Existe un algoritmo que opera en tiempo polinomial.

Teorema 5.6.1. *Dado un problema de mochila supercreciente, si existe solución para el mismo ésta es única. Además, el algoritmo SKP, explicitado en la figura 5.1, la calcula en tiempo polinomial.*

Ejemplo 5.6.1. Sea el problema de mochila dado por las condiciones:

1. $v = \langle 2, 3, 7, 15, 31 \rangle$.
2. $V = 24$

Comenzamos haciendo $\epsilon_4 = 0$ y $\epsilon_3 = 1$. En este momento debemos reemplazar 24 por 9. Seguidamente debemos hacer $\epsilon_2 = 1$ y reemplar 9 por 2. Finalmente hacemos $\epsilon_1 = 0$ y $\epsilon_0 = 1$. Ahora presentamos la upla:

$$\langle 0, 1, 1, 0, 1 \rangle$$

que en base 2 es la expresión del valor $n = 13$.

El algoritmo de cifrado para un *criptosistema de mochila* (también conocido como *criptosistema de Merkle-Hellman*) procede como sigue. En primer lugar se fija el tamaño en bits de las unidades de texto llano. Supondremos que cada unidad está representado por un número de k bits.

1. Generación de la *llave*:

   a) Se construye una sucesión supercreciente $v = \langle v_0, \ldots, v_{k-1} \rangle$

b) Se toma aleatoriamente un entero m tal que $m \geq \sum_{i=0}^{k-1} v_i$.

c) Se toma aleatoriamente $a \in \mathbb{N}^*$ tal que $a < m$ y $(a, m) = 1$.

d) Se calcula $b = a^{-1}$ mód m.

e) Se toma la sucesión $w = \langle w_0, \ldots, w_{k-1} \rangle$, donde $w_i = av_i$ mód m, para todo $0 \leq i \leq k-1$.

El usuario mantendrá secretos: v, m, a y b. Se hará público por contra w. La *llave de enciframiento* será $w = \langle w_0, \ldots, w_{k-1} \rangle$ y la de *desenciframiento* será $\langle b, m \rangle$.

2. El *enciframiento* del mensaje $M = (\epsilon_{k-1}\epsilon_{k-2}\cdots\epsilon_0)_2$ al usuario de llave pública $\langle w_0, \ldots, w_{k-1} \rangle$ será el entero

$$c = \sum_{i=0}^{k-1} \epsilon_i w_i$$

que será transmitido.

3. El *desciframiento* del mensaje c recién recibido se realiza como sigue:

a) Se calcula $V = bc$ mód m.

b) Se utiliza el algoritmo SKP aplicado a V y a la upla secreta $v = \langle v_0, \ldots, v_{k-1} \rangle$, que devolverá M.

Observación 5.6.1. La generación de la llave se podría llevar a cabo de la siguiente forma.

1. Se elige una sucesión $\langle z_0, \ldots, z_k \rangle$ de enteros positivos todos acotados por una cantidad adecuada.

2. Tomamos la sucesión finita $\langle z_0, \ldots, z_{k-1} \rangle$ definida como:

$$v_i = \begin{cases} z_0, & \text{si } i = 0 \\ z_i + \sum_{j=0}^{i-1} v_i, & \text{si } 1 \leq i \leq k-1 \end{cases}$$

3. Hacemos $m = z_k + \sum_{i=0}^{k-1} v_i$.

4. Elegimos aleatoriamente $a_0 < m$.

5. Tomamos el menor entero a tal que $a_0 < a$ y $(a, m) = 1$.

Teorema 5.6.2. *Supongamos que el mensaje* m *ha sido cifrado con la llave pública* w *obteniéndose* c. *Si a la llave pública* w *corresponde la privada* $\langle b, m \rangle$, *confeccionada a partir de los valores* a *y* v, *entonces el proceso de desciframiento del algoritmo de mochila aplicado a* c *genera* M.

Demostración. Se cumple que:

$$\begin{aligned} bc &\equiv \sum_{i=0}^{k-1} \epsilon_i b w_i \pmod{m} \\ &\equiv \sum_{i=0}^{k-1} \epsilon_i v_i \pmod{m} \end{aligned}$$

Dado que $V < m$ y $\sum_{i=0}^{k-1} \epsilon_i v_i \leq \sum_{i=0}^{k-1} v_i < m$, se tiene que $V = \sum_{i=0}^{k-1} \epsilon_i v_i$. □

Observación 5.6.2. Obsérvese que aquel intruso que quiera descifrar un mensaje C y que sólo conozca la llave pública w, se está enfrentado al problema de mochila con datos $\langle C, w \rangle$. Este problema no es un problema de mochila supercreciente —hay que evitar que lo sea, lo que representa un gran contratiempo.

Ejemplo 5.6.2. Supongamos que las unidades de nuestro texto llano son letras representadas como números de 5 bits entre $(00000)_2$ y $(11001)_2$. Supongamos que nuestro descifrador secreto está basado en la 5-upla supercreciente del ejemplo 5.6.1. Elegimos los siguientes parámetros:

1. $m = 61$,
2. $a = 17$.

entonces $b = 18$ y la llave de enciframiento es $\langle 34, 51, 58, 11, 39 \rangle$. Para enviar cifrado el mensaje WHY se deben efectuar los siguientes cálculos:

1. $W = (10110)_2$ que se traduce en $51 + 58 + 39 = 148$.
2. $H = (00111)_2$ que se traduce en $34 + 51 + 58 = 143$.
3. $Y = (11000)_2$ que se traduce en $11 + 39 = 50$.

El receptor, para leer el mensaje $\langle 148, 143, 50 \rangle$, primero multiplica por 18 y reduce módulo 61. De esta forma se obtienen los valores $\langle 41, 12, 46 \rangle$. Ahora invocamos el algoritmo SKP con en los siguientes casos:

1. $SKP(5, \langle 2, 3, 7, 15, 31 \rangle, 41; n)$, obteniéndose $n = (10110)_2$.
2. $SKP(5, \langle 2, 3, 7, 15, 31 \rangle, 12; n)$, obteniéndose $n = (00111)_2$.
3. $SKP(5, \langle 2, 3, 7, 15, 31 \rangle, 46; n)$, obteniéndose $n = (11000)_2$.

Se ha recuperado pues el texto llano.

Observación 5.6.3. Aunque la dificultad de resolver un problema de mochila es elevada, sin embargo el atacante de un sitema criptográfico que en él se basara, encontraría un problema de mochila que proviene de otro supercreciente a través de una transformación sencilla (multiplicando y reduciendo módulo). En 1982, *Shamir* encontró un algoritmo para resolver tal instancia particular del problema y que procede en tiempo polinomial. De esta forma el criptosistema original de *Merkle-Hellman* se convierte en un sistema criptográfico de llave pública poco seguro. Para reforzar su fiabilidad se han hecho algunas propuestas, entre otras la de efectuar un cifrado iterativo.

Lo cierto es que muchos criptógrafos han perdido su confianza en el criptosistema de mochila. A pesar de todo, hay un criptosistema de mochila que permanece aún sin romper y que está basado en polinomios sobre cuerpos finitos.

Capítulo 6

Algoritmos Hash Seguros

6.1. Introducción

SHA es el acrónimo de "Secure Hash Algorithm" que traducimos por "algoritmo hash seguro" (cfr [FIPS PUB 180-4]). Todos los algoritmos SHA son funciones hash iterativas y unidireccionales que pueden procesar un mensaje para producir una representación condensada llamada *resumen de mensaje*. Estos algoritmos permiten determinar la integridad de un mensaje: cualquier cambio en el mensaje, con una probabilidad altísima, dará como resultado un resumen de mensaje diferente. Esta propiedad es útil en la generación y verificación de firmas digitales y códigos de autenticación de mensajes, y en la generación de números aleatorios o bits.

Cada algoritmo se puede describir en dos etapas: preprocesamiento y cálculo hash. El preprocesamiento implica rellenar un mensaje, analizar el mensaje rellenado en bloques de m bits y establecer valores de inicialización para usar en el cálculo hash. El cálculo hash genera un cronograma de mensajes a partir del mensaje rellenado y usa ese cronograma, junto con funciones, constantes y operaciones de palabras para generar iterativamente una serie de valores hash. El valor hash final generado por el cálculo hash se utiliza para determinar el resumen del mensaje.

Los algoritmos difieren más significativamente en las fortalezas de seguridad que proporcionan para los datos que se están procesando. Los puntos fuertes de seguridad de estas funciones hash y el sistema en su conjunto cuando se utilizan con otros algoritmos criptográficos, como los algoritmos de firma digital y los códigos de autenticación de mensajes hash con clave, se pueden encontrar en: [SP 800-57. Part 1], [SP 800-57. Part 2], [SP 800-57. Part 3] y [SP 800-107].

Además, los algoritmos difieren en términos del tamaño de los bloques y las palabras de

datos que se utilizan durante el hash o los tamaños de resumen de mensajes. La Figura 1 presenta las propiedades básicas de estos algoritmos hash.

Algoritmo	Mensaje (bits)	Bloque (bits)	Palabra (bits)	Resumen (bits)
SHA-1	$< 2^{64}$	512	32	160
SHA-224	$< 2^{64}$	512	32	224
SHA-256	$< 2^{64}$	512	32	256
SHA-384	$< 2^{128}$	1024	64	384
SHA-512	$< 2^{128}$	1024	64	512
SHA-512/224	$< 2^{128}$	1024	64	224
SHA-512/256	$< 2^{128}$	1024	64	256

Figura 6.1: Propiedades de los algoritmos SHA.

Puede sorprender que los algoritmos SHA limiten el tamaño del mensaje, pero en la práctica esta limitación es de poca relevancia. Para ello consideremos el caso del libro D. Quijote de la Mancha; sus números son los de la siguiente tabla que aparece en la Figura 6.2.

n° de letras	1627392
n° de signos de puntuación	62576
n° de interrogaciones	970
n° de exclamaciones	663
n° de palabras	378119
total	2069720

Figura 6.2: Números del libro Don Quijote de la Mancha.

Según ella podemos estimar que que el número de caracteres de la obra estaría alrededor de los 2069720. Al ser codificados en binario, cada uno por 2^3 bits, resultaría en 16557760 y esta cantidad es muy inferior a 2^{64}.

6.2. Definiciones

A continuación enumeramos algunos términos útiles en este capítulo con una explicación de su significado:

Glosario de Términos y Acrónimos

- Bit; dígito binario con los valores 0 ó 1.

- Byte; grupo de 8 bits.
- FIPS; acrónimo de "Federal Information Processing Standard".
- NIST; acrónimo de "National Institute of Standards and Technology".
- SHA; acrónimo de "Secure Hash Algorithm"
- SP; acrónimo de "Special Publication"
- Word; grupo de 32 bits (4 bytes) ó 64 bits (8 bytes), dependiendo del algoritmo SHA.

Parámetros de los Algoritmos

- a, b, c,. . . , h; variables de trabajo que son palabras w-bit usadas en la computación de los valores hash, $H^{(i)}$.
- $H^{(i)}$; el i-ésimo valor hash. $H^{(0)}$ es el valor inicial hash, $H^{(N)}$ es el valor hash final y es usado para determinar el mensaje resumen (message digest).
- $H_j^{(i)}$; la j-ésima palabra del i-ésimo valor hash, donde $H_0^{(i)}$ es la palabra más a la izquierda del valor hash i.
- K_t; valor constante para ser usado en la iteración t del cálculo hash.
- k; número de ceros adjuntados a un mensaje en el paso de relleno (padding step).
- l; longitud en bits del mensaje M.
- m; número de bits en un bloque de mensaje, $M^{(i)}$.
- M; mensaje que ha de ser sometido al proceso de cálculo hash.
- $M^{(i)}$; bloque i del mensaje, con un tamaño de m bits.
- $M_j^{(i)}$; palabra j-ésima del bloque i-ésimo del mensaje, donde $M_0^{(i)}$ es la palabra más a la izquierda del bloque i de mensaje.
- n; número de bits a rotar o desplazar cuando se opera una palabra.
- N; número de bloques en el mensaje rellenado.
- T; palabra de w-bit temporal utilizada en el cálculo hash.
- w; número de bits en una palabra.
- W_t; la t-ésima w-bit palabra del mensaje extendido.

Símbolos y Operaciones de los Algoritmos

Los siguientes símbolos son usados en las especifaciones de los SHA; todos operan sobre palabras de w bits:

- $\wedge$; operación and bit a bit.
- $\vee$; operación or (inclusiva) bit a bit.
- $\oplus$; operación xor (exclusiva) bit a bit.
- $\neg$; operación de completación bit a bit.
- $+$; suma módulo 2^w.
- $<<$; operación de desplazamiento a izquierda, donde $x << n$ es obtenido descartando los n bits más a la izquierda y rellenando por la derecha con n ceros.
- $>>$; operación de desplazamiento a la derecha, donde $x >> n$ es obtenido descartando los n bits más a la derecha y rellenando por la izquierda con n ceros.
- $||$; concatenación de dos cadenas de caracteres. Por ejemplo, si $x = 11001$ e $y = 010$ entonces $x||y = 11001010$.

Las siguientes operaciones son usados en las especifaciones de los SHA:

- $\text{ROTL}^n(x)$; operación de rotar a la izquierda (desplazamiento circular a izquierda), donde x es una w-bit palabra y n un número natural tal que $0 \leq n < w$. Queda definida por la igualdad:

$$\text{ROTL}^n(x) = (x << n) \vee (x >> w - n)$$

- $\text{ROTR}^n(x)$; operación de rotar a la derecha (desplazamiento circular a derecha), donde x es una w-bit palabra y n un número natural tal que $0 \leq n < w$. Queda definida por la igualdad:

$$\text{ROTL}^n(x) = (x >> n) \vee (x << w - n)$$

- $\text{SHR}^n(x)$; operación de desplazamiento a derecha, donde x es una w-bit palabra y n un número natural tal que $0 \leq n < w$. Queda definida por la igualdad:

$$\text{SHR}^n x = (x >> n)$$

Notación

Usaremos la siguiente terminología respecto a cadenas de bits y enteros:

1. Mantendremos el convenio sobre notación hexadecimal de la tabla en la Figura 4.1.

2. Una *palabra* es una cadena (string) de w bits que puede ser representada como una secuencia de dígitos hexadecimales. Para convertir palabras en dígitos hexadecimales cada subcadena de 4 bits es convertida en su dígito hexadecimal equivalente. Por ejemplo, la palabra de 32 bits:

   1010 0001 0000 0011 1111 1110 0010 0011

   puede ser expresada como a103fe23 y la cadena de 64 bits:

   1010 0001 0000 0011 1111 1110 0010 0011 0011 0010 1110 1111 0011 0000 0001 1010

   puede ser expresada como a103fe2332ef301a. A lo largo de esta especificación, la convención "más representativo al final" (big-endian) se usa cuando se expresan palabras de 32 y 64 bits, de modo que dentro de cada palabra el bit más significativo se almacena en la posición de bit más a la izquierda.

3. Cualquier natural puede ser representado como una palabra o un par de palabras. Se necesita una representación en bits de la longitud del mensaje, l, para la técnicas de relleno. Un natural entre 0 y $2^{32}-1$, ambos inclusive, puede ser representado por una palabra de 32 bits. Los cuatro dígitos menos significativos del natural están representados por el dígito hexadecimal más a la derecha en la representación de la palabra. Por ejemplo, el natural

   $$\begin{aligned} 291 &= 256 + 32 + 2 + 1 \\ &= 2^8 + 2^5 + 2^1 + 2^0 \end{aligned}$$

   es representado por la palabra en hexadecimal 00000123. Lo mismo vale para los naturales entre 0 y $2^{64} - 1$ ambos inclusive, los cuales pueden ser representados por una palabra de 64 bits.

   Si z es un número natural tal que $0 \leq z < 2^{64}$, entonces existen números naturales x e y tales que $0 \leq x, y < 2^{32}$ y $z = 2^{32}x + y$. Así, z puede ser representado por el par $\langle x, y \rangle$, es decir, por un par de palabras de 32 bits. Esta propiedad es usada por SHA-1, SHA-224 y SHA-256.

Si z es un número natural tal que $0 \leq z < 2^{128}$, entonces existen números naturales x e y tales que $0 \leq x, y < 2^{64}$ y $z = 2^{64}x + y$. Así, z puede ser representado por el par $\langle x, y \rangle$, es decir, por un par de palabras de 64 bits. Esta propiedad es usada por SHA-384, SHA-512/224 y SHA-512/256.

4. En los algoritmos de hash seguro el tamaño del bloque de mensaje, m bits, depende del algoritmo:

   a) Para SHA-1, SHA-224 y SHA-256, cada bloque de mensaje es de 512 bits, que está representado por una sucesión de 16 palabras de 32 bits.

   b) Para SHA-384, SHA-512/224 y SHA-512/256, cada bloque de mensaje es de 1024 bits, que está representado por una sucesión de 16 palabras de 64 bits.

Operaciones Sobre Palabras

En los algoritmos SHA son aplicadas las siguientes operación a las palabras de w bits. En los algoritmos SHA-1, SHA-224 y SHA-256 w vale 32 y en los algoritmos SHA-384, SHA-512, SHA-512/224 y SHA-512/256 w vale 64.

1. Operaciones lógicas bit a bit: $\wedge$, $\vee$, $\oplus$ y $\neg$.

2. Suma módulo 2^w. Si las palabras x e y representan naturales X e Y respectivamente, entonces se obtiene:

   $$Z = (X + Y) \text{ mód } 2^w$$

   y entonces Z es convertido en una palabra, digamos z, y por último decimos que z es el valor de $x + y$ por definición.

3. La operación de *desplazmiento a derecha* (right shift) $\text{SHR}^n(x)$, donde x es una palabra de w bits y n es un número natural tal que $0 \leq n < w$. Su definición es:

   $$\text{SHR}^n(x) = x >> n$$

   Esta operación es usada en los algoritmos SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 y SHA-512/256.

4. La operación de *rotación a derecha* (rotate right o circular right shift) $\text{ROTR}^n(x)$, donde x es una palabra de w bits y n es un número natural tal que $0 \leq n < w$. Su definición es:

   $$\text{ROTR}^n(x) = (x >> n) \vee (x << (w - n))$$

   Así pues, $\text{ROTR}^n(x)$ es equivalente a un desplazamiento circular de x en n posiciones a la derecha. Esta operación es usada en los algoritmos SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 y SHA-512/256.

5. La operación de *rotación a izquierda* (rotate left o circular left shift) $\text{ROTL}^n(x)$, donde x es una palabra de w bits y n es un número natural tal que $0 \leq n < w$. Su definición es:

$$\text{ROTL}^n(x) = (x << n) \vee (x >> (w - n))$$

Así pues, $\text{ROTL}^n(x)$ es equivalente a un desplazamiento circular de x en n posiciones a la izquierda. Esta operación es usada en el algoritmo SHA-1.

Lema 6.2.1. *Para toda palabra x de w bits y para todo número natural n tal que $0 \leq n < w$ se cumplen las siguientes igualdades:*

$$\text{ROTL}^n(x) = \text{ROTR}^{w-n}(x)$$
$$\text{ROTR}^n(x) = \text{ROTL}^{w-n}(x)$$

Demostración. Es consecuencia inmediata de las definiciones. □

6.3. Funciones

Los diversos algoritmos SHA utilizan funciones que definiremos en esta sección. En algunos casos intervienen las siguientes sobre ternas de palabras de 32 bits:

$$\text{Ch}(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z) \tag{6.1}$$
$$\text{Parity}(x, y, z) = x \oplus y \oplus z \tag{6.2}$$
$$\text{Maj}(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \tag{6.3}$$

En la literatura se hace alusión a las constantes usadas en los algoritmos SHA; no obstante, cualquier constante es una función de ariedad 0.

Funciones de SHA-1

SHA-1 utiliza la sucesión de funciones $\{f_t\}_{0 \leq t < 80}$, cada una de las cuales opera sobre las palabras x, y y z de 32 bits y produce una nueva palabra de 32 bits. Para todo $0 \leq t < 80$, f_t queda definida por la siguiente igualdad:

$$f_t(x,y,z) = \begin{cases} Ch(x,y,z), & 0 \leq t < 20 \\ Parity(x,y,z), & 20 \leq t < 40 \\ Maj(x,y,z), & 40 \leq t < 60 \\ Parity(x,y,z), & 60 \leq t < 80 \end{cases} \tag{6.4}$$

Constantes de SHA-1

SHA-1 utiliza la sucesión de constantes $\{K_t\}_{0 \leq t < 80}$, dada por las siguientes igualdades:

$$K_t = \begin{cases} 5a827999 & 0 \leq t < 20 \\ 6ed9eba1 & 20 \leq t < 40 \\ 8f1bbcdc & 40 \leq t < 60 \\ ca62c1d6 & 60 \leq t < 80 \end{cases} \tag{6.5}$$

Funciones de SHA-224 y SHA-256

Las funciones que intervienen en los algoritmos SHA-224 y SHA-256 son $Ch(x,y,z)$ y $Maj(x,y,z)$, además de:

$$\sum_0^{\{256\}}(x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x) \tag{6.6}$$
$$\sum_1^{\{256\}}(x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x) \tag{6.7}$$
$$\sigma_0^{\{256\}}(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x) \tag{6.8}$$
$$\sigma_1^{\{256\}}(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x) \tag{6.9}$$

Constantes de SHA-224 y SHA-256

SHA-224 y SHA-256 usan la misma sucesión de 64 constantes, $\{K_t^{\{256\}}\}_{0 \leq t < 64}$, cada una de las cuales es una palabra de 32 bits. Estas palabras representan los primeros 32 bits de la parte racional de las raíces cúbicas de los primeros sesenta y cuatro números primos.

Ejemplo 6.3.1. Se tiene que:

$$\sqrt[3]{2} = 1{,}259921049894873164767210607278228350570251464701507980081\ldots$$

y que:

$$(\sqrt[3]{2})_2 = 1{,}010000101000101000101111100110001101011100101000101011100010001 0\ldots$$

En efecto:

```
sage: a=2
sage: b = a.n(digits=20)
sage: b.nth_root(3)
1.2599210498948731648
sage: 0.25992104989487316488*2
0.51984209978974633
sage: 0.51984209978974633 *2 # 0
1.0396841995794927
sage: 0.0396841995794927*2 # 1
0.0793683991589854
sage: 0.0793683991589854 * 2 # 0
0.158736798317971
sage: 0.158736798317971 *2 # 0
0.317473596635942
sage: 0.317473596635942*2 # 0
0.634947193271884
sage: 0.634947193271884*2 # 0
1.26989438654377
sage: 0.26989438654377 * 2 # 1
0.539788773087540
sage: 0.539788773087540 *2 # 0
1.07957754617508
sage: 0.07957754617508 * 2 # 1
0.159155092350160
sage: 0.159155092350160 * 2 # 0
0.318310184700320
sage: 0.318310184700320 * 2 # 0
0.636620369400640
sage: 0.636620369400640 * 2 # 0
1.27324073880128
sage: 0.27324073880128 * 2 # 1
0.546481477602560
sage: 0.546481477602560 * 2 # 0
1.09296295520512
sage: 0.09296295520512 * 2 # 1
0.185925910410240
```

La anterior secuencia de números hace que podamos establecer que $\sqrt[3]{2}$ tenga en la parte decimal de su expresión binaria la siguiente secuencia de dígitos:

$$010000101000101\ldots$$

Así pues:

$$\begin{aligned}(0100)_2 &= (4)_{16}\\ (0010)_2 &= (2)_{16}\\ (1000)_2 &= (8)_{16}\\ (1010)_2 &= (a)_{16}\\ (0010)_2 &= (2)_{16}\\ (1111)_2 &= (f)_{16}\\ (1001)_2 &= (9)_{16}\\ (1000)_2 &= (8)_{16}\end{aligned}$$

de donde K_1 será la palabra de 32 bits `428a2f98`.

En hexadecimal estas palabras constantes, extraídas de la raíz cúbica de los sesenta y cuatro primeros números primos, son de izquierda a derecha:

```
428a2f98 71374491 b5c0fbcf e9b5dba5 3956c25b 59f111f1 923f82a4 ab1c5ed5
d807aa98 12835b01 243185be 550c7dc3 72be5d74 80deb1fe 9bdc06a7 c19bf174
e49b69c1 efbe4786 0fc19dc6 240ca1cc 2de92c6f 4a7484aa 5cb0a9dc 76f988da
983e5152 a831c66d b00327c8 bf597fc7 c6e00bf3 d5a79147 06ca6351 14292967
27b70a85 2e1b2138 4d2c6dfc 53380d13 650a7354 766a0abb 81c2c92e 92722c85
a2bfe8a1 a81a664b c24b8b70 c76c51a3 d192e819 d6990624 f40e3585 106aa070
19a4c116 1e376c08 2748774c 34b0bcb5 391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3
748f82ee 78a5636f 84c87814 8cc70208 90befffa a4506ceb bef9a3f7 c67178f2
```

Funciones de SHA-384, SHA-512, SHA-512/224 y SHA-512/256

SHA-384, SHA-512, SHA-512/224 y SHA-512/256 usan seis funciones lógicas six logical, cada una de las cuales opera sobre palabras de 64 bits representadas por x, y y z; el resultado de cada función es una nueva palabra de 64 bits. Además de las funciones $Ch(x,y,z)$ y $Maj(x,y,z)$ vienen involucradas estas otras:

$$\sum_0^{\{512\}}(x) = \text{ROTR}^{28}(x) \oplus \text{ROTR}^{34}(x) \oplus \text{ROTR}^{39}(x) \tag{6.10}$$

$$\sum_1^{\{512\}}(x) = \text{ROTR}^{14}(x) \oplus \text{ROTR}^{18}(x) \oplus \text{ROTR}^{41}(x) \tag{6.11}$$

$$\sigma_0^{\{512\}}(x) = \text{ROTR}^{1}(x) \oplus \text{ROTR}^{8}(x) \oplus \text{SHR}^{7}(x) \tag{6.12}$$

$$\sigma_1^{\{512\}}(x) = \text{ROTR}^{19}(x) \oplus \text{ROTR}^{61}(x) \oplus \text{SHR}^{6}(x) \tag{6.13}$$

Constantes de SHA-384, SHA-512, SHA-512/224 y SHA-512/256

SHA-384, SHA-512, SHA-512/224 and SHA-512/256 usan la misma sucesión de ochenta constantes de palabras de 64 bits, $\{K_t^{\{512\}}\}_{0 \le t < 80}$. Estas palabras representan los primeros 64 bits de la parte racional de las raíces cúbicas de los primeros ochenta números primos y son de izquierda a derecha:

```
428a2f98d728ae22 7137449123ef65cd b5c0fbcfec4d3b2f e9b5dba58189dbbc
3956c25bf348b538 59f111f1b605d019 923f82a4af194f9b ab1c5ed5da6d8118
d807aa98a3030242 12835b0145706fbe 243185be4ee4b28c 550c7dc3d5ffb4e2
72be5d74f27b896f 80deb1fe3b1696b1 9bdc06a725c71235 c19bf174cf692694
e49b69c19ef14ad2 efbe4786384f25e3 0fc19dc68b8cd5b5 240ca1cc77ac9c65
2de92c6f592b0275 4a7484aa6ea6e483 5cb0a9dcbd41fbd4 76f988da831153b5
983e5152ee66dfab a831c66d2db43210 b00327c898fb213f bf597fc7beef0ee4
c6e00bf33da88fc2 d5a79147930aa725 06ca6351e003826f 142929670a0e6e70
27b70a8546d22ffc 2e1b21385c26c926 4d2c6dfc5ac42aed 53380d139d95b3df
650a73548baf63de 766a0abb3c77b2a8 81c2c92e47edaee6 92722c851482353b
a2bfe8a14cf10364 a81a664bbc423001 c24b8b70d0f89791 c76c51a30654be30
d192e819d6ef5218 d69906245565a910 f40e35855771202a 106aa07032bbd1b8
19a4c116b8d2d0c8 1e376c085141ab53 2748774cdf8eeb99 34b0bcb5e19b48a8
391c0cb3c5c95a63 4ed8aa4ae3418acb 5b9cca4f7763e373 682e6ff3d6b2b8a3
748f82ee5defb2fc 78a5636f43172f60 84c87814a1f0ab72 8cc702081a6439ec
90befffa23631e28 a4506cebde82bde9 bef9a3f7b2c67915 c67178f2e372532b
ca273eceea26619c d186b8c721c0c207 eada7dd6cde0eb1e f57d4f7fee6ed178
06f067aa72176fba 0a637dc5a2c898a6 113f9804bef90dae 1b710b35131c471b
28db77f523047d84 32caab7b40c72493 3c9ebe0a15c9bebc 431d67c49c100d4c
4cc5d4becb3e42b6 597f299cfc657e2a 5fcb6fab3ad6faec 6c44198c4a475817
```

6.4. Procesado Previo del Mensaje

El procesado previo del mensaje consta de tres fases:

- Relleno de completación (padding) del mensaje M.
- Ajuste del mensaje a un patrón de bloques de mensaje (parsing the message into message blocks).
- Colocación (setting) de los valores hash iniciales, $H(0)$.

Relleno de Completación

El propósito de este relleno es asegurar que el mensaje relleno tiene una longitud múltiplo de 512 ó 1024 bits, dependiendo del algoritmo. El relleno sobre un mensaje puede ser hecho antes de que el cálculo comience o en cualquier otro momento del cálculo hash, pero antes de procesar los bloques que contendrían el relleno.

El relleno presenta diferencia según el algoritmo hash.

SHA-1, SHA-224 y SHA-256

Supongamos que la longitud del mensaje M es l bits; en este caso se cumple $0 < l < 2^{64}$ y supongamos que $(l)_2$ es la secuencia de caracteres de 64 bits que representa a l en expresión binaria. En segundo lugar es preciso encontrar el menor número natural k que cumple:

$$l + 1 + k \equiv 448 \pmod{512}$$

Con estos valores:

- añadimos en yuxtaposición el bit 1 al final del mensaje.
- añadimos en yuxtaposición k bits 0.
- añadimos en yuxtaposición los 64 bits que provienen de expresar l en base 2.

El mensaje M relleno para la completación será representado por $P_1(M)$.

Ejemplo 6.4.1. Supongamos que M es 'abc'. En la condificación UTF-8 el carácter 'a' (resp. 'b', 'c') tiene la representación hexadecimal 61 (resp. 62, 63), es decir, 01100001 (resp. 01100010, 01100011). Entonces $l = 24$, de forma que:

$$\begin{aligned} 24 + 1 + k \equiv 448 \pmod{512} \text{ sii } k &\equiv 448 - 25 \pmod{512} \\ \text{sii } k &\equiv 423 \pmod{512} \end{aligned}$$

de modo que $k = 423$. Por otra parte $(24)_2 = 00\overset{59)}{\ldots}011000$. En definitiva, el mensaje relleno por completación será:

$$\underbrace{01100001}_{'a'}\ \underbrace{01100010}_{'b'}\ \underbrace{01100011}_{'c'}\ 1\ \overbrace{00\ldots0}^{423}\ \overbrace{00\ldots0\underbrace{11000}_{l=24}}^{64}$$

Teorema 6.4.1. *Para todo mensaje no vacío* M*, si* $\mathrm{length}(P_1(M))$ *es la longitud de* $P_1(M)$ *entonces:*

$$\mathrm{length}(P_1(M)) \equiv 0 \pmod{512}$$

Demostración. Se tiene que:

$$\begin{aligned} l + 1 + ((448 - l - 1) \bmod 512) + 64 &\equiv (l + 65 + ((447 - l) \bmod 512)) \pmod{512} \\ &\equiv (l + 65 + 447 - l) \pmod{512} \\ &\equiv 512 \pmod{512} \\ &\equiv 0 \pmod{512} \end{aligned}$$

es decir,

$$\mathrm{length}(P_1(M)) \equiv 0 \pmod{512}$$

□

SHA-384, SHA-512, SHA-512/224 y SHA-512/256

Supongamos que la longitud del mensaje M es l bits; en este caso se cumple $0 < l < 2^{128}$ y supongamos que $(l)_2$ es la secuencia de caracteres de 128 bits que representa a l en expresión binaria. En segundo lugar es preciso encontrar el menor número natural k que cumple:

$$l + 1 + k \equiv 896 \pmod{1024}$$

Con estos valores:

- añadimos en yuxtaposición el bit 1 al final del mensaje.

- añadimos en yuxtaposición k bits 0.
- añadimos en yuxtaposición los 128 bits que provienen de expresar l en base 2.

El mensaje M relleno para la completación será representado por $P_3(M)$.

Ejemplo 6.4.2. Supongamos que M es 'abc'. En la condificación UTF-8 el carácter 'a' (resp. 'b', 'c') tiene la representación hexadecimal 61 (resp. 62, 63), es decir, 01100001 (resp. 01100010, 01100011). Entonces $l = 24$, de forma que:

$$\begin{aligned} 24 + 1 + k \equiv 896 \pmod{1024} \text{ sii } k &\equiv 896 - 25 \pmod{1024} \\ \text{sii } k &\equiv 871 \pmod{1024} \end{aligned}$$

de modo que $k = 871$. Por otra parte $(24)_2 = 00\overset{123)}{\ldots}011000$. En definitiva, el mensaje relleno por completación será:

$$\underbrace{01100001}_{'a'}\ \underbrace{01100010}_{'b'}\ \underbrace{01100011}_{'c'}\ 1\ \overbrace{00\ldots0}^{871}\ \overbrace{00\ldots0\underbrace{11000}_{l=24}}^{128}$$

Teorema 6.4.2. *Para todo mensaje no vacío* M*, si* $\mathrm{length}(P_3(M))$ *es la longitud de* $P_3(M)$ *entonces:*

$$\mathrm{length}(P_3(M)) \equiv 0 \pmod{1024}$$

Demostración. Se tiene que:

$$\begin{aligned} l + 1 + ((896 - l - 1) \bmod 1024) + 128 &\equiv (l + 129 + ((895 - l) \bmod 1024)) \pmod{1024} \\ &\equiv (l + 129 + 895 - l) \pmod{1024} \\ &\equiv 1024 \pmod{1024} \\ &\equiv 0 \pmod{1024} \end{aligned}$$

es decir,

$$\mathrm{length}(P_1(M)) \equiv 0 \pmod{1024}$$

□

Ajuste del Mensaje a un Patrón

El mensaje una vez relleno para su completación debe ser ajustado a N bloques de m-bits cada uno; el número m depende del algoritmo hash y puede valer 512 o 1024.

SHA-1, SHA-224 y SHA-256

Dado el mensaje no vacío M, sea su completación $P_1(M)$. El mensaje $P_1(M)$ es dividido en en N bloques de 512 bits, donde N es el número natural que cumple (cfr. Teorema 6.4.1):

$$length(P_1(M)) = 512N$$

Esto proporciona la sucesión de bloques de 512 bits $\{M^{(k)}\}_{1\leq k\leq N}$. Ahora bien, cada bloque $M^{(k)}$ puede ser dividido en 16 bloques de 32 bits; así pues, para todo $1 \leq k \leq N$ surge una nueva sucesión, a saber $\{M_j^{(k)}\}_{0\leq j\leq 15}$, compuesta de bloques de 32 bits.

SHA-384, SHA-512, SHA-512/224 y SHA-512/256

Dado el mensaje no vacío M, sea su completación $P_3(M)$. El mensaje $P_3(M)$ es dividido en en N bloques de 1024 bits, donde N es el número natural que cumple (cfr. Teorema 6.4.2):

$$length(P_3(M)) = 1024N$$

Esto proporciona la sucesión de bloques de 1024 bits $\{M^{(k)}\}_{1\leq k\leq N}$. Ahora bien, cada bloque $M^{(k)}$ puede ser dividido en 16 bloques de 64 bits; así pues, para todo $1 \leq k \leq N$ surge una nueva sucesión, a saber $\{M_j^{(k)}\}_{0\leq j\leq 15}$, compuesta de bloques de 64 bits.

Abusando de la notación, llamaremos también $P_1(M)$ (resp. $P_3(M)$) a la sucesión

$$\{\{M_j^{(k)}\}_{0\leq j\leq 15}\}_{1\leq k\leq N}$$

Inicialización de los Valores Hash

Antes de que comience la computación de los algoritmos hash deben de ser instanciados los valores hash iniciales. El tamaño y número de palabras en $H^{(0)}$ dependen del tamaño del mensaje resumen.

Para los algoritmos SHA-1, SHA-224, SHA-256, SHA-384 y SHA-512 el valor hash inicial, $H^{(0)}$, consta de las palabras de 32 bits (expresadas en hexadecimal) que refiere la tabla de la Figura 6.3

Los algoritmos SHA-512/t hacen depender la inicialización de $H^{(0)}$ del valor de t; este valor es un número natural y cumple $0 \leq t < 512$, pero debe ser distinto de 384. En el nombre del algoritmo, si damos a t el valor 256 —como mero ejemplo— escribiremos "SHA-512/256" y no "SHA-512/0256". Especialmente aprobados para tomar parte en

los algoritmos hash son los valores de t: 224 y 256. Para otros valores de t puede consultar el documento [SP 800-107]

La geración de $H^{(0)}$ para inicializar los algoritmos SHA-512/t es por medio de la función "SHA-512/t IV Generation Function", cuyo procedimiento de cálculo está detallado en el esquema de la Figura 6.5. En la línea 02. del pseudocódigo de la Figura 6.5, a5a5a5a5a5a5a5a5 está expresado en hexadecimal y en la línea 03. queremos indicar que SHA-512 opera con $H^{(0)'}$ en lugar del $H^{(0)}$ predeterminado para dicho algoritmo SHA-512.

A continuación examinamos el ejemplo $t = 256$.

H(0)	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
$H_0^{(0)}$	67452301	c1059ed8	6a09e667	cbbb9d5dc1059ed8	6a09e667f3bcc908
$H_1^{(0)}$	efcdab89	367cd507	bb67ae85	629a292a367cd507	bb67ae8584caa73b
$H_2^{(0)}$	98badcfe	3070dd17	3c6ef372	9159015a3070dd17	3c6ef372fe94f82b
$H_3^{(0)}$	10325476	f70e5939	a54ff53a	152fecd8f70e5939	a54ff53a5f1d36f1
$H_4^{(0)}$	c3d2e1f0	ffc00b31	510e527f	67332667ffc00b31	510e527fade682d1
$H_5^{(0)}$		68581511	9b05688c	8eb44a8768581511	9b05688c2b3e6c1f
$H_6^{(0)}$		64f98fa7	1f83d9ab	db0c2e0d64f98fa7	1f83d9abfb41bd6b
$H_7^{(0)}$		befa4fa4	5be0cd19	47b5481dbefa4fa4	5be0cd19137e2179

Figura 6.3: $H^{(0)}$ para: SHA-1, SHA-224, SHA-256, SHA-384 y SHA-512

input: t en su expresión decimal, sujeto a lo especificado.
output: $H^{(0)}$ como cadena de 512 bits o como lista de 8 palabras de 64 bits.
procedure SHA−512(M)
    begin
01.) for i = 1 to 7:
02.) $H_i^{(0)'} = H_i^{(0)} \oplus a5a5a5a5a5a5a5a5$
03.) $H^{(0)} = SHA-512('SHA-512/t'; H^{(0)'})$
04.) return $H^{(0)}$
    end

Figura 6.4: SHA-512/t IV Generation Function o generación de H(0) para SHA-512/t.

$$
\begin{aligned}
H^{(0)} = [&0x6a09e667f3bcc908,\\
&0xbb67ae8584caa73b,\\
&0x3c6ef372fe94f82b,\\
&0xa54ff53a5f1d36f1,\\
&0x510e527fade682d1,\\
&0x9b05688c2b3e6c1f,\\
&0x1f83d9abfb41bd6b,\\
&0x5be0cd19137e2179]\\
H^{(0)'} = [&0x6a09e667f3bcc908 \oplus 0xa5a5a5a5a5a5a5a5,\\
&0xbb67ae8584caa73b \oplus 0xa5a5a5a5a5a5a5a5,\\
&0x3c6ef372fe94f82b \oplus 0xa5a5a5a5a5a5a5a5,\\
&0xa54ff53a5f1d36f1 \oplus 0xa5a5a5a5a5a5a5a5,\\
&0x510e527fade682d1 \oplus 0xa5a5a5a5a5a5a5a5,\\
&0x9b05688c2b3e6c1f \oplus 0xa5a5a5a5a5a5a5a5,\\
&0x1f83d9abfb41bd6b \oplus 0xa5a5a5a5a5a5a5a5,\\
&0x5be0cd19137e2179 \oplus 0xa5a5a5a5a5a5a5a5]
\end{aligned}
$$

Esto da como resultado para $H^{(0)'}$ lo siguiente:

$$
\begin{aligned}
H^{(0)'} = [&\text{'cfac43c256196cad'},\\
&\text{'1ec20b20216f029e'},\\
&\text{'99cb56d75b315d8e'},\\
&\text{'00ea509ffab89354'},\\
&\text{'f4abf7da08432774'},\\
&\text{'3ea0cd298e9bc9ba'},\\
&\text{'ba267c0e5ee418ce'},\\
&\text{'fe4568bcb6db84dc'}]
\end{aligned}
$$

variable $s =' SHA{-}512/256'$

Ahora ejecutamos el algoritmo normal SHA-512 sobre el string `'SHA-512/256'`, aunque con su $H^{(0)}$ original reemplazado por $H^{(0)'}$, que habrá sido generado previamente.

variable $H^{(0)} = SHA{-}512(s; H^{(0)'})$

Éste acaba en 8 palabras de 64 bits cada una, que serán usadas como $H^{(0)}$ en nuestro algoritmo SHA-512/t. Por ejemplo: SHA-512 con $H^{(0)'}$ en el papel del $H^{(0)}$ (valores predeterminados para el algoritmo SHA-512, cfr. Figura 6.3) sobre el string 'SHA-512/256' proporciona el siguiente string (en hex):

```
22312194FC2BF72C9F555FA3C84C64C22393B86B6F53B151963877195940EABD
96283EE2A88EFFE3BE5E1E25538639922B0199FC2C85B8AA0EB72DDC81C52CA2
```

o estas 8 palabras (también en hex):

```
22312194FC2BF72C
9F555FA3C84C64C2
2393B86B6F53B151
963877195940EABD
96283EE2A88EFFE3
BE5E1E2553863992
2B0199FC2C85B8AA
0EB72DDC81C52CA2
```

Estos son los valores $H_i^{(0)}$ para SHA-512/256 especificados en [FIPS 180-4], sección 5.3.6.2, pág. 17.

La Figura 6.5 resume los resultados del cáculo de los parámetros iniciales $H^{(0)}$ para $t = 224$ y $t = 256$ por medio de la ejecución de SHA-512/t IV Generation Function con $t = 224$ y $t = 256$.

$H^{(0)}$	SHA-512/224	SHA-512/256
$H_0^{(0)}$	8C3D37C819544DA2	22312194FC2BF72C
$H_1^{(0)}$	73E1996689DCD4D6	9F555FA3C84C64C2
$H_2^{(0)}$	1DFAB7AE32FF9C82	2393B86B6F53B151
$H_3^{(0)}$	679DD514582F9FCF	963877195940EABD
$H_4^{(0)}$	0F6D2B697BD44DA8	96283EE2A88EFFE3
$H_5^{(0)}$	77E36F7304C48942	BE5E1E2553863992
$H_6^{(0)}$	3F9D85A86A1D36C8	2B0199FC2C85B8AA
$H_7^{(0)}$	1112E6AD91D692A1	0EB72DDC81C52CA2

Figura 6.5: $H^{(0)}$ para SHA-512/224 y SHA-512/256

6.5. Algoritmo Hash Seguro SHA-1

SHA-1 puede ser usado para elaborar el mensaje resumen de 160 bits H, a partir de un mensaje M que en bits tenga longitud l, donde $0 \leq l < 2^{64}$. La explicación detallada del algoritmo es como explica el contenido de la Figura 6.6. Sea tenido en cuenta que aquí la operación $+$ es realizada módulo 2^{32}.

input: Un mensaje M no vacío de longitud l bits, $0 \leq l < 2^{64}$.
output: Un mensaje resumen H de 160 bits.
procedure SHA−1(M)
begin
01.) $\{\{M_j^{(k)}\}_{0\leq j\leq 15}\}_{1\leq k\leq N} = P_1(M)$
02.) for i = 1 to N:
03.) $W_t = \begin{cases} M_t^{(i)}, & 0 \leq t \leq 15 \\ ROTL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}), & 16 \leq t \leq 79 \end{cases}$
04.) for j = 0 to 4:
05.) $a_j = H_j^{(i-1)}$
06.) for t = 0 to 79:
07.) $T = ROTL^5(a_0) + f_t(a_1, a_2, a_3) + a_4 + K_t + W_t$
08.) $a_4 = a_3$
09.) $a_3 = a_2$
10.) $a_2 = ROTL^{30}(a_1)$
11.) $a_1 = a_0$
12.) $a_0 = T$
13.) for j = 0 to 4:
14.) $H_j^{(i)} = a_j + H_j^{(i-1)}$
15.) $H = H_0^{(N)} || H_1^{(N)} || H_2^{(N)} || H_3^{(N)} || H_4^{(N)}$
16.) return H
end

Figura 6.6: Algoritmo SHA-1.

El método de cálculo hash de SHA-1 explicado en la Figura 6.6 supone que la extensión del mensaje $\{W_i\}_{0\leq i\leq 79}$ está implementado por medio de un array de ochenta palabras de 32 bits. Ello es eficiente desde el punto de vista del tiempo de ejecución, por minimizarlo, puesto que las direcciones de $W_{t-3}, \ldots, W_{t-16}$ son fácilmente calculadas (en 03.). No obstante, si la memoria es limitada, una alternativa es ver $\{W_t\}$ como una cola circular que puede ser implementada por medio de un array de palabras de 32-bits, W_0,

$W_1, \ldots, W_{15}$. El método alternativo que describimos ahora (cfr. Figura 6.7) ahorra el almacenamiento de sesenta y cuatro palabras de 32 bits, pero es probable que alargue el tiempo de ejecución debido al incremento en la complejidad de la dirección de cálculo para los W_t (en 06.). Para este método alternativo, sea $MASK = 0000000f$ (en hex).

input: Un mensaje M no vacío de longitud l bits, $0 \leq l < 2^{64}$.
output: Un mensaje resumen H de 160 bits.
procedure $SHA{-}1{-}Alt(M)$
begin
01.) $\{\{M_j^{(k)}\}_{0\leq j\leq 15}\}_{1\leq k\leq N} = P_1(M)$
02.) for $i = 1$ to N:
03.) for $t = 0$ to 15:
04.) $W_t = M_t^{(i)}$
05.) for $j = 0$ to 4:
06.) $a_j = H_j^{(i-1)}$
07.) for $t = 0$ to 79:
08.) $s = t \wedge MASK$
09.) if $t \geq 16$ then:
10.) $W_s = ROTL^1(W_{(s+13)\wedge MASK} \oplus W_{(s+8)\wedge MASK} \oplus W_{(s+2)\wedge MASK} \oplus W_s)$
11.) $T = ROTL^5(a_0) + f_t(a_1, a_2, a_3) + a_4 + K_t + W_s$
12.) $a_4 = a_3$
13.) $a_3 = a_2$
14.) $a_2 = ROTL^{30}(a_1)$
15.) $a_1 = a_0$
16.) $a_0 = T$
17.) for $j = 0$ to 4:
18.) $H_j^{(i)} = a_j + H_j^{(i-1)}$
19.) $H = H_0^{(N)} || H_1^{(N)} || H_2^{(N)} || H_3^{(N)} || H_4^{(N)}$
20.) return H
end

Figura 6.7: Algoritmo SHA-1 alternativo.

6.6. Algoritmo Hash Seguro SHA-256

SHA-256 puede ser usado para elaborar el mensaje resumen de 256 bits H, a partir de un mensaje M que en bits tenga longitud l, donde $0 \leq l < 2^{64}$. La explicación detallada

del algoritmo es como explica el contenido de la Figura 6.8. Sea tenido en cuenta que aquí la operación + es realizada módulo 2^{32}.

input: Un mensaje M no vacío de longitud l bits, $0 \leq l < 2^{64}$.
output: Un mensaje resumen H de 256 bits.
procedure SHA−256(M)
begin
01.) $\{\{M_j^{(k)}\}_{0\leq j\leq 15}\}_{1\leq k\leq N} = P_1(M)$
02.) for $i = 1$ to N:
03.) $$W_t = \begin{cases} M_t^{(i)}, & 0 \leq t \leq 15 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16}, & 16 \leq t \leq 63 \end{cases}$$
04.) for $j = 0$ to 7:
05.) $a_j = H_j^{(i-1)}$
06.) for $t = 0$ to 63:
07.) $T_1 = h + \sum_1^{\{256\}}(a_4) + Ch(a_4, a_5, a_6) + K_t^{\{256\}} + W_t$
08.) $T_2 = \sum_0^{\{256\}}(a_0) + Maj(a_0, a_1, a_2)$
09.) $a_7 = a_6$
10.) $a_6 = a_5$
11.) $a_5 = a_4$
12.) $a_4 = a_3 + T_1$
13.) $a_3 = a_2$
14.) $a_2 = a_1$
15.) $a_1 = a_0$
16.) $a_0 = T_1 + T_2$
17.) for $j = 0$ to 7:
18.) $H_j^{(i)} = a_j + H_j^{(i-1)}$
19.) $H = H_0^{(N)}||H_1^{(N)}||H_2^{(N)}||H_3^{(N)}||H_4^{(N)}||H_5^{(N)}||H_6^{(N)}||H_7^{(N)}$
20.) return H
end

Figura 6.8: Algoritmo SHA-256.

6.7. Algoritmo Hash Seguro SHA-224

SHA-224 puede ser usado para elaborar el mensaje resumen de 224 bits H, a partir de un mensaje M que en bits tenga longitud l, donde $0 \leq l < 2^{64}$. El algoritmo es el de SHA-256 explicado en la Figura 6.8 con exactamente las siguientes salvedades:

1. Los valores iniciales $H^{(0)}$ son los que se le atribuyen al algoritmo SHA-224 en la Figura 6.3

2. El mensaje resumen es obtenido truncando el mensaje hash final H a sus 224 bits más a la izquierda, es decir, el algoritmo devuelve:

$$H_0^{(N)} || H_1^{(N)} || H_2^{(N)} || H_3^{(N)} || H_4^{(N)} || H_5^{(N)} || H_6^{(N)}$$

6.8. Algoritmo Hash Seguro SHA-512

SHA-512 puede ser usado para elaborar el mensaje resumen de 512 bits H, a partir de un mensaje M que en bits tenga longitud l, donde $0 \leq l < 2^{128}$. La explicación detallada del algoritmo es como explica el contenido de la Figura 6.9. Sea tenido en cuenta que aquí la operación $+$ es realizada módulo 2^{64}.

6.9. Algoritmo Hash Seguro SHA-384

SHA-384 puede ser usado para elaborar el mensaje resumen de 384 bits H, a partir de un mensaje M que en bits tenga longitud l, donde $0 \leq l < 2^{128}$. El algoritmo es el de SHA-512 explicado en la Figura 6.9 con exactamente las siguientes salvedades:

1. Los valores iniciales $H^{(0)}$ son los que se le atribuyen al algoritmo SHA-384 en la Figura 6.3

2. El mensaje resumen es obtenido truncando el mensaje hash final H a sus 384 bits más a la izquierda, es decir, el algoritmo devuelve:

$$H_0^{(N)} || H_1^{(N)} || H_2^{(N)} || H_3^{(N)} || H_4^{(N)} || H_5^{(N)}$$

6.10. Algoritmo Hash Seguro SHA-512/224

SHA-512/224 puede ser usado para elaborar el mensaje resumen de 224 bits H, a partir de un mensaje M que en bits tenga longitud l, donde $0 \leq l < 2^{128}$. El algoritmo es el de SHA-512 explicado en la Figura 6.9 con exactamente las siguientes salvedades:

1. Los valores iniciales $H^{(0)}$ son los que se le atribuyen al algoritmo SHA-512/224 en la Figura 6.5

input: Un mensaje M no vacío de longitud l bits, $0 \leq l < 2^{128}$.
output: Un mensaje resumen H de 512 bits.
procedure SHA−512(M)
begin
01.) $\{\{M_j^{(k)}\}_{0\leq j\leq 15}\}_{1\leq k\leq N} = P_3(M)$
02.) for $i = 1$ to N:
03.) $W_t = \begin{cases} M_t^{(i)}, & 0 \leq t \leq 15 \\ \sigma_1^{\{512\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{512\}}(W_{t-15}) + W_{t-16}, & 16 \leq t \leq 79 \end{cases}$
04.) for $j = 0$ to 7:
05.) $a_j = H_j^{(i-1)}$
06.) for $t = 0$ to 79:
07.) $T_1 = h + \sum_1^{\{512\}}(a_4) + Ch(a_4, a_5, a_6) + K_t^{\{512\}} + W_t$
08.) $T_2 = \sum_0^{\{512\}}(a_0) + Maj(a_0, a_1, a_2)$
09.) $a_7 = a_6$
10.) $a_6 = a_5$
11.) $a_5 = a_4$
12.) $a_4 = a_3 + T_1$
13.) $a_3 = a_2$
14.) $a_2 = a_1$
15.) $a_1 = a_0$
16.) $a_0 = T_1 + T_2$
17.) for $j = 0$ to 7:
18.) $H_j^{(i)} = a_j + H_j^{(i-1)}$
19.) $H = H_0^{(N)}||H_1^{(N)}||H_2^{(N)}||H_3^{(N)}||H_4^{(N)}||H_5^{(N)}||H_6^{(N)}||H_7^{(N)}$
20.) return H
end

Figura 6.9: Algoritmo SHA-512.

2. El mensaje resumen es obtenido truncando el mensaje hash final H a sus 224 bits más a la izquierda.

6.11. Algoritmo Hash Seguro SHA-512/256

SHA-512/256 puede ser usado para elaborar el mensaje resumen de 256 bits H, a partir de un mensaje M que en bits tenga longitud l, donde $0 \leq l < 2^{128}$. El algoritmo es el de SHA-512 explicado en la Figura 6.9 con exactamente las siguientes salvedades:

1. Los valores iniciales $H^{(0)}$ son los que se le atribuyen al algoritmo SHA-512/256 en la Figura 6.5
2. El mensaje resumen es obtenido truncando el mensaje hash final H a sus 256 bits más a la izquierda.

Capítulo 7

Certificados Digitales y Aplicaciones

7.1. Introducción

En criptografía X.509 es un estándar que define el formato para los certificados de clave pública. Los certificados X.509 se utilizan en muchos protocolos de Internet incluido TLS/SSL que es la base de HTTPS, el protocolo seguro para navegar por la web; también es usado en aplicaciones fuera de línea como firmas electrónicas. Un certificado X.509 contiene una clave pública y una identidad (un nombre de host, una organización o un individuo), y está firmado por una autoridad de certificación (CA) o autofirmado. Cuando un certificado es firmado por una autoridad de certificación de confianza, o validado por otros medios, alguien que posee ese certificado puede confiar en la clave pública que contiene para establecer comunicaciones seguras con otra parte o validar documentos firmados digitalmente por la clave privada correspondiente.

Una autoridad de certificación (AC o CA, en inglés) es una entidad que emite certificados digitales para su uso por terceros; es un ejemplo de un tercero de confianza. Las AC son características en muchos esquemas de infraestructuras de clave pública (PKI). Existen muchas CA comerciales que cobran por sus servicios. Instituciones y gobiernos pueden tener sus propias CA y también existen CA gratuitas.

X.509 también define listas de revocación de certificados, que son un medio para distribuir información sobre certificados que una autoridad de firma ha considerado inválidos, así como un algoritmo de validación de ruta de certificación, que permite que los certificados sean firmados por certificados de CA intermedios, que son a su vez firmados por otros certificados, llegando finalmente a un ancla de confianza que es una entidad autorizada para la cual se asume la confianza sin ser ésta deriva hacia instancia superior alguna.

En la arquitectura X.509, un certificado raíz (root certificate) sería el ancla de confianza de la que se deriva toda la cadena de confianza. Para hacer posible cualquier validación de ruta de certificado adicional, el ancla de confianza debe estar en posesión de la parte que de antemano confía.

La mayoría de los sistemas operativos proporcionan una lista integrada de certificados raíz autofirmados para actuar como anclas de confianza para las aplicaciones. El navegador web Firefox también proporciona su propia lista de anclas de confianza. El usuario final de un sistema operativo o navegador web está confiando implícitamente en el funcionamiento correcto de ese software, y el fabricante del software a su vez está delegando la confianza para ciertas operaciones criptográficas a las autoridades de certificación responsables de los certificados raíz.

Los anclajes de confianza basados en hardware incluyen regiones de memoria valladas o circuitos separados que se supone que son seguros.

X.509 está definido por el sector de Normalización de la Unión Internacional de Telecomunicaciones (UIT-T) y se basa en ASN.1, otro estándar del UIT-T.

X.509 se emitió inicialmente el 3 de julio de 1988 y se inició en asociación con el estándar X.500. Asume un estricto sistema jerárquico de autoridades de certificación (CA) para emitir los certificados. Esto contrasta con la web de modelos de confianza, como PGP, donde cualquiera (no sólo CA especiales) puede firmar y, por lo tanto, dar fe de la validez de los certificados clave de otros. La versión 3 de X.509 incluye la flexibilidad para admitir otras topologías además de cadenas, como son puentes y mallas. Se puede usar en una red de confianza entre pares, similar a OpenPGP, pero rara vez se usó de esa manera a partir de 2004.

El sistema X.500 sólo ha sido implementado por naciones soberanas para propósitos de cumplimiento de tratados de intercambio de información de identidad estatal, y el grupo de trabajo de Infraestructura de Clave Pública (X.509) de IETF —acrónimo del nombre de la organización “Internet Engineering Task Force”, o PKIX, ha adaptado el estándar a la organización más flexible del Internet. De hecho, el término certificado X.509 generalmente se refiere al certificado PKIX de IETF y al perfil CRL del estándar de certificado X.509 v3, como se especifica en RFC 5280 comúnmente llamado PKIX para infraestructura de clave pública (X.509).

7.2. Certificación

En el sistema X.509, una organización que quiere un certificado firmado solicita uno a través de una solicitud de firma de certificado (CSR, acrónimo de "certificate signing request"). En los sistemas de infraestructura de clave pública (PKI, acrónimo de "public key infrastructure") una solicitud de firma de certificado (CSR) es un mensaje enviado por un solicitante a una autoridad de certificación para solicitar un certificado de identidad digital. Por lo general, contiene la clave pública para la cual se debe emitir el certificado, que identifica información (como un nombre de dominio) y protección de integridad (por ejemplo, una firma digital). El formato más común para las CSR es la especificación PKCS#10 y otro es la clave pública firmada y el formato Challenge SPKAC generado por algunos navegadores web.

Para hacer esto primero genera un par de claves, manteniendo en secreto la clave privada y usándola para firmar la CSR. Ésta contiene información que identifica al solicitante, la clave pública del solicitante que se utiliza para verificar la firma de la CSR y el Nombre Distinguido (DN) para el que se destina el certificado. La CSR puede estar acompañada de otras credenciales o pruebas de identidad requeridas por la autoridad de certificación.

Los certificados raíz (root certificates) de confianza de una organización se pueden distribuir a todos los empleados para que puedan usar el sistema PKI de la compañía. Los navegadores como Internet Explorer, Firefox, Opera, Safari y Chrome vienen con un conjunto predeterminado de certificados raíz preinstalados, entonces los certificados SSL de las principales autoridades certificadoras funcionarán instantáneamente; en efecto, los desarrolladores de los navegadores determinan qué CA son terceros de confianza para los usuarios de los navegadores. Por ejemplo, Firefox proporciona un archivo CSV y/o HTML que contiene una lista de CA incluidas. A pesar de que estos certificados raíz pueden borrarse o deshabilitarse, es muy raro que los usuarios lo hagan.

X.509 y RFC 5280 también incluyen estándares para implementaciones de listas de revocación de certificados (CRL, acrónimo de "certificate revocation list"). Otra forma aprobada por IETF para verificar la validez de un certificado es el Protocolo de estado de certificado en línea (OCSP, acrónimo de "Online Certificate Status Protocol"). Firefox 3 habilita la comprobación de OCSP de forma predeterminada, al igual que las versiones de Windows de al menos Vista y posteriores.

Estructura de un Certificado

La estructura prevista por las normas está expresada en un lenguaje formal, la "Abstract Syntax Notation One" (ASN.1).

La estructura de un certificado digital X.509 v3 es la siguiente:

- Certificado
  - Número de versión
  - Número de serie
  - ID de algoritmo de firma
  - Nombre del emisor
  - Periodo de validez
    - No antes
    - No después de
  - Nombre del asunto
  - Información de Clave Pública del asunto
    - Algoritmo de clave pública
    - Asunto de la Clave Pública.
  - Identificador único del emisor (opcional)
  - Identificador único del asunto (opcional)
  - Extensiones (opcional)
    - ...
- Algoritmo de firma del certificado
- Firma de certificado

Extensiones de nombre de archivo de certificado

Existen varias extensiones de nombre de archivo de uso común para los certificados X.509. Desafortunadamente, algunas de estas extensiones también se utilizan para otros datos, como las claves privadas.

- `.pem` - (abreviatura de "Privacy-enhanced Electronic Mail", Correo electrónico con privacidad mejorada) Certificado DER (Distinguished Encoding Rules) codificado en Base64, encerrado entre "`----- BEGIN CERTIFICATE -----`z"`----- END CERTIFICATE ----`
- `.cer`, `.crt`, `.der` - generalmente en formato DER binario, pero los certificados codificados en Base64 también son comunes (ver `.pem` arriba).

- `.p7b`, `.p7c` - Estructura SignedData PKCS#7 sin datos, sólo certificado(s) o CRL(s)
- `.p12` - PKCS#12, puede contener certificado(s) (público) y claves privadas (protegido con contraseña).
- `.pfx`: PFX, predecesor de PKCS#12 (generalmente contiene datos en formato PKCS#12, por ejemplo, con archivos PFX generados en IIS)

PKCS#7 es un estándar para firmar o cifrar datos (oficialmente llamados .envolventes"). Como el certificado es necesario para verificar los datos firmados, es posible incluirlos en la estructura SignedData. Un archivo `.P7C` es una estructura SignedData degenerada, sin datos para firmar.

PKCS#12 evolucionó del estándar de intercambio de información personal (PFX) y se utiliza para intercambiar objetos públicos y privados en un solo archivo.

7.3. Cadenas de Certificados y Certificaciones Cruzadas

Una cadena de certificados (ver el concepto equivalente de "ruta de certificación" definida por RFC 5280) es una lista de certificados (generalmente comenzando con un certificado de entidad final) seguido de uno o más certificados de CA (generalmente el último es un certificado autofirmado), con las siguientes propiedades:

1. El Emisor de cada certificado (excepto el último) coincide con el Subject (asunto) del próximo certificado en la lista.
2. Se supone que cada certificado (excepto el último) debe estar firmado por la clave secreta correspondiente al siguiente certificado de la cadena (es decir, la firma de un certificado se puede verificar utilizando la clave pública contenida en el siguiente certificado).
3. El último certificado de la lista es un ancla de confianza: un certificado en el que confía porque le fue entregado por algún procedimiento confiable.

Las cadenas de certificados se utilizan para verificar que la clave pública (PK) contenida en un certificado de destino (el primer certificado de la cadena) y otros datos contenidos

en ella pertenecen efectivamente a su sujeto. Para determinar esto, la firma en el certificado de destino se verifica usando el PK contenido en el siguiente certificado, cuya firma se verifica usando el siguiente certificado, y así sucesivamente hasta que se alcanza el último certificado en la cadena. Como el último certificado es un ancla de confianza, alcanzarlo con éxito demostrará que se puede confiar en el certificado de destino.

La descripción en el párrafo anterior es una vista simplificada sobre el proceso de validación de la ruta de certificación según lo definido por RFC 5280, que involucra verificaciones adicionales, como verificar fechas de validez en certificados, buscar CRL, etc.

Al examinar cómo se crean y validan las cadenas de certificados es importante tener en cuenta que un certificado concreto puede ser parte de cadenas de certificados muy diferentes (todas ellas válidas). Esto se debe a que se pueden generar varios certificados de CA para el mismo sujeto y clave pública, pero se pueden firmar con diferentes claves privadas (de diferentes CA o diferentes claves privadas de la misma CA). Por lo tanto, aunque un solo certificado X.509 puede tener sólo un emisor y una firma de CA, se puede vincular válidamente a más de un certificado, creando cadenas de certificados completamente diferentes. Esto es crucial para la certificación cruzada entre PKI y otras aplicaciones. Vea los siguientes ejemplos.

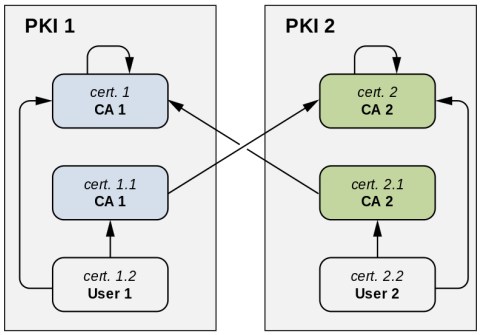
En estos diagramas:

- Cada cuadro representa un certificado, con su Asunto (Subject) en negrita.
- A $\rightarrow$ B significa "A está firmado por B" (o, más precisamente, "A está firmado por la clave secreta correspondiente a la clave pública contenida en B").
- Los certificados con el mismo color (que no son blancos/transparentes) contienen la misma clave pública.

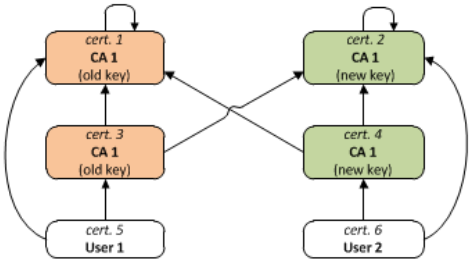
- Ejemplo 1 (Figura 7.1a): Certificación cruzada a nivel de la Autoridad de certificación raíz (CA) entre dos PKI (acrónimo de "Public Key Infrastructure", infraestructura de clave pública):

  Para gestionar que PKI 1 confíe en los certificados de usuario existentes en PKI 2 (como "Usuario 2"), CA1 genera un certificado (cert2.1) que contiene la clave pública de CA2. Ahora, tanto cert2 como cert2.1 (en verde) tienen el mismo asunto y clave pública, por lo que hay dos cadenas válidas para cert2.2 (Usuario 2): "cert2.2 $\rightarrow$ cert2" y "cert2.2 $\rightarrow$ cert2.1 $\rightarrow$ cert1".

  De manera similar, CA2 puede generar un certificado (cert1.1) que contiene la clave pública de CA1 para que PKI 2 confíe en los certificados de usuario existentes en PKI 1 (como "Usuario 1").



(a) Ejemplo 1: Certificaciones cruzadas entre dos PKI



(b) Ejemplo 2: Renovación de certificado de CA

Figura 7.1: Referencia cruzadas entre PKI.

- Comprender la construcción de la ruta de certificación (PDF). Foro PKI. Septiembre de 2002. "Para permitir una transición elegante del antiguo par de claves de firma al nuevo par de claves de firma, la CA debe emitir un certificado que contenga la antigua clave pública firmada por la nueva clave de firma privada y un certificado que contenga la nueva clave pública firmada por la antigua clave de firma privada. Ambos certificados son autoemitidos, pero ninguno es autofirmado. Tenga en cuenta que estos se suman a los dos certificados autofirmados (uno antiguo y uno nuevo)".

  Dado que tanto cert1 como cert3 contienen la misma clave pública (la anterior), existen dos cadenas de certificados válidas para cert5: "cert5 → cert1" y "cert5 → cert3 → cert2", y de forma análoga para cert6. Esto permite que los certificados de usuario antiguos (como cert5) y los nuevos certificados (como cert6) puedan ser confiados indistintamente por una parte que tenga el nuevo certificado raíz de CA o el antiguo como ancla de confianza durante la transición a las nuevas claves de CA.

7.4. Ejemplos de Certificados X.509

Mostramos un ejemplo de un certificado X.509 decodificado que fue utilizado por wikipedia.org y varios otros sitios web de Wikipedia. Fue emitido por GlobalSign, como se indica en el campo Issuer (Emisor). Su campo Subject (Asunto) describe Wikipedia como una organización, y su campo Subject Alternative Name describe los nombres de host para los que podría usarse. El campo Subject Public Key Info contiene una clave pública ECDSA, mientras que la firma en la parte inferior fue generada por la clave privada RSA de GlobalSign.

Certificado de Entidad Final

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            10:e6:fc:62:b7:41:8a:d5:00:5e:45:b6
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=BE, O=GlobalSign nv-sa, CN=GlobalSign Organization Validation CA - SHA256 - G2
        Validity
            Not Before: Nov 21 08:00:00 2016 GMT
            Not After : Nov 22 07:59:59 2017 GMT
        Subject: C=US, ST=California, L=San Francisco, O=Wikimedia Foundation, Inc., CN=*.wikipedia.org
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
            pub:
                    00:c9:22:69:31:8a:d6:6c:ea:da:c3:7f:2c:ac:a5:
                    af:c0:02:ea:81:cb:65:b9:fd:0c:6d:46:5b:c9:1e:
                    9d:3b:ef
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature, Key Agreement
            Authority Information Access:
                CA Issuers - URI:http://secure.globalsign.com/cacert/gsorganizationvalsha2g2r1.crt
                OCSP - URI:http://ocsp2.globalsign.com/gsorganizationvalsha2g2
            X509v3 Certificate Policies:
                Policy: 1.3.6.1.4.1.4146.1.20
                  CPS: https://www.globalsign.com/repository/
                Policy: 2.23.140.1.2.2
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 CRL Distribution Points:
                Full Name:
                  URI:http://crl.globalsign.com/gs/gsorganizationvalsha2g2.crl
            X509v3 Subject Alternative Name:
                DNS:*.wikipedia.org, DNS:*.m.mediawiki.org, DNS:*.m.wikibooks.org,
                DNS:*.m.wikidata.org, DNS:*.m.wikimedia.org, DNS:*.m.wikimediafoundation.org,
                DNS:*.m.wikinews.org, DNS:*.m.wikipedia.org, DNS:*.m.wikiquote.org,
                DNS:*.m.wikisource.org, DNS:*.m.wikiversity.org, DNS:*.m.wikivoyage.org,
                DNS:*.m.wiktionary.org, DNS:*.mediawiki.org, DNS:*.planet.wikimedia.org,
                DNS:*.wikibooks.org, DNS:*.wikidata.org, DNS:*.wikimedia.org,
                DNS:*.wikimediafoundation.org, DNS:*.wikinews.org, DNS:*.wikiquote.org,
                DNS:*.wikisource.org, DNS:*.wikiversity.org, DNS:*.wikivoyage.org,
                DNS:*.wiktionary.org, DNS:*.wmfusercontent.org, DNS:*.zero.wikipedia.org,
                DNS:mediawiki.org, DNS:w.wiki, DNS:wikibooks.org, DNS:wikidata.org,
                DNS:wikimedia.org, DNS:wikimediafoundation.org, DNS:wikinews.org,
                DNS:wikiquote.org, DNS:wikisource.org, DNS:wikiversity.org, DNS:wikivoyage.org,
                DNS:wiktionary.org, DNS:wmfusercontent.org, DNS:wikipedia.org
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
            X509v3 Subject Key Identifier:
                28:2A:26:2A:57:8B:3B:CE:B4:D6:AB:54:EF:D7:38:21:2C:49:5C:36
            X509v3 Authority Key Identifier:
                keyid:96:DE:61:F1:BD:1C:16:29:53:1C:C0:CC:7D:3B:83:00:40:E6:1A:7C

    Signature Algorithm: sha256WithRSAEncryption
         8b:c3:ed:d1:9d:39:6f:af:40:72:bd:1e:18:5e:30:54:23:35:
         ...
```

Para validar este certificado de entidad final, se necesita un certificado intermedio que coincida con su Identificador de clave de emisor y autoridad:

Issuer	C=BE, O=GlobalSign nv-sa, CN=GlobalSign Organization Validation CA - SHA256 - G2
Authority Key Identifier	96:DE:61:F1:BD:1C:16:29:53:1C:C0:CC:7D:3B:83:00:40:E6:1A:7C

En una conexión TLS, un servidor configurado correctamente proporcionaría el intermedio como parte del protocolo de enlace. Sin embargo, también es posible recuperar el certificado intermedio mediante la obtención de la URL "CA Issuers" del certificado de entidad final.

Intermediate certificate (certificado intermedio)

Éste es un ejemplo de un certificado intermedio que pertenece a una autoridad de certificación. Este certificado firmó el certificado de entidad final anterior y fue firmado por el certificado raíz a continuación. Tenga en cuenta que el campo subject de este certificado intermedio coincide con el campo del emisor del certificado de entidad final que firmó. Además, el campo "subject key identifier" en el intermedio coincide con el campo "authority key identifier" en el certificado de entidad final.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            04:00:00:00:00:01:44:4e:f0:42:47
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=BE, O=GlobalSign nv-sa, OU=Root CA, CN=GlobalSign Root CA
        Validity
            Not Before: Feb 20 10:00:00 2014 GMT
            Not After : Feb 20 10:00:00 2024 GMT
        Subject: C=BE, O=GlobalSign nv-sa, CN=GlobalSign Organization Validation CA - SHA256 - G2
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:c7:0e:6c:3f:23:93:7f:cc:70:a5:9d:20:c3:0e:
                    ...
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Key Usage: critical
                Certificate Sign, CRL Sign
            X509v3 Basic Constraints: critical
                CA:TRUE, pathlen:0
            X509v3 Subject Key Identifier:
                96:DE:61:F1:BD:1C:16:29:53:1C:C0:CC:7D:3B:83:00:40:E6:1A:7C
            X509v3 Certificate Policies:
                Policy: X509v3 Any Policy
                  CPS: https://www.globalsign.com/repository/

            X509v3 CRL Distribution Points:

                Full Name:
                  URI:http://crl.globalsign.net/root.crl
```

```
            Authority Information Access:
                OCSP - URI:http://ocsp.globalsign.com/rootr1

            X509v3 Authority Key Identifier:
                keyid:60:7B:66:1A:45:0D:97:CA:89:50:2F:7D:04:CD:34:A8:FF:FC:FD:4B

    Signature Algorithm: sha256WithRSAEncryption
         46:2a:ee:5e:bd:ae:01:60:37:31:11:86:71:74:b6:46:49:c8:
         ...
```

Certificado Root

Éste es un ejemplo de un certificado raíz autofirmado que representa una autoridad de certificación. Sus campos issuer (emisor) y subject (sujeto) son los mismos, y su firma se puede validar con su propia clave pública. La validación de la cadena de confianza tiene que terminar aquí. Si el programa de validación tiene este certificado raíz en su almacén de confianza, el certificado de entidad final puede considerarse confiable para su uso en una conexión TLS. De lo contrario, el certificado de entidad final se considera no confiable.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            04:00:00:00:00:01:15:4b:5a:c3:94
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=BE, O=GlobalSign nv-sa, OU=Root CA, CN=GlobalSign Root CA
        Validity
            Not Before: Sep  1 12:00:00 1998 GMT
            Not After : Jan 28 12:00:00 2028 GMT
        Subject: C=BE, O=GlobalSign nv-sa, OU=Root CA, CN=GlobalSign Root CA
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:da:0e:e6:99:8d:ce:a3:e3:4f:8a:7e:fb:f1:8b:
                    ...
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Key Usage: critical
                Certificate Sign, CRL Sign
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Subject Key Identifier:
                60:7B:66:1A:45:0D:97:CA:89:50:2F:7D:04:CD:34:A8:FF:FC:FD:4B
    Signature Algorithm: sha1WithRSAEncryption
         d6:73:e7:7c:4f:76:d0:8d:bf:ec:ba:a2:be:34:c5:28:32:b5:
         ...
```

Apéndice A

Congruencias Cuadráticas

A.1. Previos

Teorema A.1.1. *Sean* $a, b, c \in \mathbb{Z}$ *y supongamos que* $(a, b) = 1$. *Si* $a|bc$ *entonces* $a|c$.

Demostración. Si $(a, b) = 1$ entonces existen $r, s \in \mathbb{Z}$ tales que $ar + bs = 1$ y por tanto, $c = acr + bcs$. Si $a|bc$, entonces $a|bcs$. Como evidentemente $a|acr$, se deduce que $a|acr + bcs$, o sea, $a|c$. □

A.2. Congruencias Cuadráticas

Lema A.2.1. *Sea* p *un número primo impar. Si* $p \mid x^2 - 1$ *entonces se cumple una de las siguientes alternativas:*

1. $p \mid (x + 1)$ *y* $p \nmid (x - 1)$
2. $p \nmid (x + 1)$ *y* $p \mid (x - 1)$

Demostración. Al ser $x^2 - 1 = (x + 1)(x - 1)$, si $p \mid x^2 - 1$ entonces $p \mid (x + 1)$ o $p \mid (x - 1)$. Supongamos que $p \mid (x + 1)$ y $p \mid (x - 1)$; entonces p sería un divisor de $(x+1)-(x-1) = 2$ y por tanto par, lo cual es imposible. Por tanto, si $p \mid x^2-1$ entonces divide exactamente a uno de los números $x + 1$ o $x - 1$. □

Lema A.2.2. *Sea* p *un número primo impar y* e *un número natural no nulo. Entonces la ecuación*

$$x^2 \equiv 1 \pmod{p^e} \tag{A.1}$$

tiene exactamente dos soluciones y éstas son $x \equiv \pm 1 \pmod{p}$.

Demostración. Es claro que $(\pm 1)^2 \equiv 1 \pmod{p^e}$. Como $1 \not\equiv -1 \pmod{p^e}$, hay al menos dos soluciones. El resto de la demostración está dedicado a demostrar que sólo hay esas dos. Si $p^e \mid x^2 - 1$ entonces $p \mid x^2 - 1$ y por tanto p no divide a uno de los factores $(x-1)$ y $(x+1)$ (cfr. Lema A.2.1) y el otro será divisible por p^e. Si $p^e \mid (x+1)$, entonces $x \mid -1 \pmod{p^e}$ y si $p^e \mid (x-1)$ entonces $x \equiv 1 \pmod{p^e}$. En definitiva, las únicas soluciones son $x \equiv \pm 1 \pmod{p^e}$. □

Ejercicio A.2.1. Encontrar las soluciones de $x^2 \equiv 1 \pmod{2^4}$ a partir de las de $x^2 \equiv 1 \pmod{2^3}$ razonando por la expresión de las mismas en base 2.

Solución. Las soluciones de $x^2 \equiv 1 \pmod{2^3}$ son:

- $1 = 001$
- $3 = 011$
- $5 = 101$
- $7 = 111$

Cada solución de $x^2 \equiv 1 \pmod{2^4}$ debe dar una de $x^2 \equiv 1 \pmod{2^3}$ truncándola por la tercer posición, pues si $2^4 \mid (x^2 - 1)$ entonces $2^3 \mid (x^2 - 1)$. Por tanto las soluciones de $x^2 \equiv 1 \pmod{2^4}$ están entre:

- 0001
- 1001
- 0011
- 1011
- 0101
- 1101
- 0111

- 1111

Se puede comprobar que **exactamente son soluciones**:

- $0001 = 1 \equiv 1 \pmod{2^4}$
- $1001 = 9 \equiv (1 + 2^3) \pmod{2^4}$
- $0111 = 7 \equiv -(1 + 2^3) \pmod{2^4}$
- $1111 = 15 \equiv -1 \pmod{2^4}$

mientras que **no** aportan una **solución**:

- $0011 = 3 \equiv -(1 + 2^2 + 2^3) \pmod{2^4}$
- $1011 = 11 \equiv -(1 + 2^2) \pmod{2^4}$
- $0101 = 5 \equiv (1 + 2^2) \pmod{2^4}$
- $1101 = 13 \equiv (1 + 2^2 + 2^3) \pmod{2^4}$

□

Lema A.2.3. *Sea e un número natural no nulo. La ecuación:*

$$x^2 \equiv 1 \pmod{2^e} \tag{A.2}$$

tiene las siguientes soluciones:

1. $e = 1$; $x \equiv 1 \pmod 2$ *(una única solución)*
2. $e = 2$; $x \equiv 1 \pmod{2^2}$ *y* $x \equiv 3 \pmod{2^2}$ *(es decir* $x \equiv \pm 1 \pmod{2^2}$*, exactamente dos soluciones)*
3. $3 \leq e$, $x \equiv \pm 1 \pmod{2^e}$ *o* $x \equiv \pm(1 + 2^{e-1}) \pmod{2^e}$ *(es decir, cuatro soluciones).*

Demostración. Los casos $e = 1$ y $e = 2$ pueden ser demostrados por simple constatación. Para la afirmación sobre el caso $3 \leq e$ razonamos por inducción. Para $n = 3$ el resultado es cierto ya que las soluciones son $x \equiv 1, 3, 5, 7 \pmod{2^3}$, es decir $x \equiv \pm 1, \pm 5 \pmod{2^2}$, exactamente cuatro soluciones. Supongamos que $3 \leq e$ y que el resultado es cierto para e; demostrémoslo para $e + 1$. Supongamos que x es una solución de $x^2 \equiv 1$

$(\text{mód } 2^{e+1})$. Entonces $2^{e+1} \mid (x^2-1)$ y, por tanto, $2^e \mid (x^2-1)$. Así pues $x \bmod 2^e$ será una de las soluciones enumeradas en el caso e. Considerando la expresión en base 2 de x, debe x cumplir una de las ocho posibilidades siguientes:

$$\begin{aligned} x &\equiv \pm 1 \pmod{2^{e+1}} \\ x &\equiv \pm(1+2^e) \pmod{2^{e+1}} \\ x &\equiv \pm(1+2^{e-1}) \pmod{2^{e+1}} \\ x &\equiv \pm(1+2^{e-1}+2^e) \pmod{2^{e+1}} \end{aligned}$$

No obstante, veremos que sólo

$$x \equiv \pm 1 \pmod{2^{e+1}} \qquad \text{o} \qquad x \equiv \pm(1+2^e) \pmod{2^{e+1}}$$

son soluciones. Las primera es obviamente solución, para la segunda obsérvese:

$$\begin{aligned} (1+2^e)^2 &\equiv 1 + 2\cdot 2^e + 2^{2e} \pmod{2^{e+1}} \\ &\equiv 1 + 2^{e+1} + 2^{2e} \pmod{2^{e+1}} \\ &\equiv 1 + 2^{e+1}(1+2^{e-1}) \pmod{2^{e+1}} \\ &\equiv 1 \pmod{2^{e+1}} \end{aligned}$$

Para la tercera obsérvese que:

$$\begin{aligned} (1+2^{e-1})^2 &\equiv 1 + 2\cdot 2^{e-1} + 2^{2(e-1)} \pmod{2^{e+1}} \\ &\equiv 1 + 2^e + 2^{2e-2} \pmod{2^{e+1}} \\ &\equiv 1 + 2^e + 2^{e+1}2^{e-3} \pmod{2^{e+1}} \qquad (3 \le e) \\ &\equiv 1 + 2^e \pmod{2^{e+1}} \\ &\not\equiv 1 \pmod{2^{e+1}} \end{aligned}$$

Análogamente para la cuarta, obsérvese que:

$$\begin{aligned} (1+2^{e-1}+2^e)^2 &\equiv 1 + 2^e + 2^{2e-2} + 2^{2e} + 2^{e+1}(1+2^{e-1}) \pmod{2^{e+1}} \\ &\equiv 1 + 2^e + 2^{e+1}(2^{e-3} + 2^{e-1} + 1 + 2^{e-1}) \pmod{2^{e+1}} \\ &\equiv 1 + 2^e + 2^{e+1}(1 + 2^e + 2^{e-3}) \pmod{2^{e+1}} \\ &\equiv 1 + 2^e \pmod{2^{e+1}} \\ &\not\equiv 1 \pmod{2^{e+1}} \end{aligned}$$

□

Lema A.2.4. *Sean m_1 y m_2 números naturales mayores que uno y $m = m_1 m_2$. Si $(m_1, m_2) = 1$ entonces son equivalentes las siguientes afirmaciones:*

1. *x_0 es solución de $x^2 \equiv 1 \pmod{m}$*

2. x_0 *es solución del sistema*

$$\begin{cases} x^2 & \equiv 1 \pmod{m_1} \\ x^2 & \equiv 1 \pmod{m_2} \end{cases} \tag{A.3}$$

Demostración. Supongamos que x_0 es solución de $x^2 \equiv 1 \pmod{m}$, es decir, $m \mid x_0^2 - 1$. Para todo $i \in \{1, 2\}$, como $m_i \mid m$ entonces $m_i \mid x_0^2 - 1$ y por tanto $m_i \mid x_0^2 - 1$, o sea, x_0 es solución del sistema A.3. Recíprocamente, dicho sistema tiene solución por ser $(m_1, m_2) = 1$ y si x_0 es una de ellas, entonces $m_i \mid x^2 - 1$ para todo $i \in \{1, 2\}$. Como $(m_1, m_2) = 1$, por el Teorema A.1.1 sabemos que $m \mid x_0^2 - 1$. Así pues, x_0 es solución de $x^2 \equiv 1 \pmod{m}$. □

Lema A.2.5. *Sea* m *un número natural mayor que* 1 *y* $\prod_{i=1}^{r} p_i^{e_i}$ *su única (salvo el orden) factorización como producto de potencias de números primos (para todo* $1 \leq i \leq r$, p_i *es primo y* $0 < e_i$*). Son equivalentes las siguientes afirmaciones:*

1. x_0 *es solución de* $x^2 \equiv 1 \pmod{m}$

2. x_0 *es solución del sistema*

$$x^2 \equiv 1 \pmod{p_i^{e_i}}, \qquad 1 \leq i \leq r \tag{A.4}$$

Demostración. Si $r = 1$, no hay nada que demostrar pues las afirmaciones cuya equivalencia pretendemos demostrar son exactamente la misma. Supongamos que $1 < r$, que $m = \prod_{i=1}^{r} p_i^{e_i}$ tiene r factores primos y que el resultado es cierto para todo número mayor que 1 con $r - 1$ factores primos. Entonces sea $m_1 = \prod_{i=1}^{r-1} p_i^{e_i}$ y $m_2 = p_r^{e_r}$. Está claro que $(m_1, m_2) = 1$ y entonces, por el Lema A.2.4, x_0 es solución de $x^2 \equiv 1 \pmod{m}$ sii x_0 es solución del sistema A.3. Pero por la hipótesis de inducción, x_0 es solución de $x^2 \equiv 1 \pmod{m_1}$ sii x_0 es solución del sistema $x^2 \equiv 1 \pmod{p_i^{e_i}}$, $1 \leq i \leq r - 1$. De esto se deduce que el resultado es cierto para cualquier número m que siendo mayor que 1 tenga r factores en su única factorización. □

Teorema A.2.6. *Sea* m *es un número natural superior a* 1, *y* $\prod_{i=1}^{r} p_i^{e_i}$ *su única (salvo el orden) factorización como producto de potencias de números primos (para todo* $1 \leq i \leq r$, p_i *es primo y* $0 < e_i$*). Sea* S_i *el conjunto de soluciones de la congruencia* $x^2 \equiv 1 \pmod{p_i^{e_i}}$*. Para todo número natural* x_0*, son equivalentes las siguientes afirmaciones:*

1. x_0 *es solución de* $x^2 \equiv 1 \pmod{m}$.

2. *existe* $\langle a_i \rangle_{1 \leq i \leq r} \in \prod_{i=1}^{r} S_i$ *tal que* x_0 *es solución del sistema*

$$x \equiv a_i \pmod{p_i^{e_i}}, \qquad 1 \leq i \leq r \tag{A.5}$$

Demostración. Supongamos que x_0 es solución de $x^2 \equiv 1 \pmod{m}$. Por el Lema A.2.5 sabemos que entonces x_0 es solución de $x^2 \equiv 1 \pmod{p_i^{e_i}}$ para todo $1 \leq i \leq r$. Por el Lema A.2.2 y el Lema A.2.3 sabemos que para todo $1 \leq i \leq r$ existirá $a_i \in S_i$ tal que x_0 satisfaga $x \equiv a_i \pmod{p_i^{e_i}}$. Recíprocamente, si existe $\langle a_i \rangle_{1 \leq i \leq r} \in \prod_{i=1}^{r} S_i$ tal que x_0 es solución del sistema A.5, entonces por los lemas A.2.2 y A.2.3 sabemos que x_0 es solución sel sistema A.4 y por tanto de la ecuación $x^2 \equiv 1 \pmod{m}$, según el Lema A.2.5 de nuevo. □

Corolario A.2.7. *Sea m es un número natural superior a 1, y $\prod_{i=1}^{r} p_i^{e_i}$ su única (salvo el orden) factorización como producto de potencias de números primos (para todo $1 \leq i \leq r$, p_i es primo y $0 < e_i$). Sea S_i el conjunto de soluciones de la congruencia $x^2 \equiv 1 \pmod{p_i^{e_i}}$. El número de soluciones x_0 de $x^2 \equiv 1 \pmod{m}$ cumpliendo $1 \leq x_0 < m$ coincide con el cardinal del conjunto $\prod_{i=1}^{r} S_i$.*

Demostración. Supongamos que $\langle a_i \rangle_{1 \leq i \leq r}, \langle b_i \rangle_{1 \leq i \leq r} \in \prod_{i=1}^{r} S_i$, que x_1 es una solución de $x \equiv a_i \pmod{p_i^{e_i}}$, $1 \leq i \leq r$, que x_1 es una solución de $x \equiv a_i \pmod{p_i^{e_i}}$, $1 \leq i \leq r$ y que $x_1 \equiv x_2 \pmod{m}$. Entonces para todo $1 \leq i \leq r$, $x_1 \equiv x_2 \pmod{p_i^{e_i}}$ de donde para todo $1 \leq i \leq r$, $a_i \equiv b_i \pmod{p_i^{e_i}}$ y por tanto $\langle a_i \rangle_{1 \leq i \leq r} = \langle b_i \rangle_{1 \leq i \leq r}$. La afirmación es entonces consecuencia directa del Teorema A.2.6. □

Corolario A.2.8. *Si m es un número natural superior a 1 y $m = 2^{e_0}(\prod_{i=1}^{r} p_i^{e_i})$, donde para todo $1 \leq i \leq r$ se cumple que p_i es un número primo impar y $0 \leq e_i$, entonces la ecuación $x^2 \equiv 1 \pmod{m}$ tiene exáctamente 2^{r+e} soluciones distíntas módulo m siendo:*

$$e = \begin{cases} 0 & , \text{ si } e_0 \in \{0, 1\} \\ 1 & , \text{ si } e_0 = 2 \\ 2 & , \text{ si } e_0 \geq 3 \end{cases}$$

Demostración. Es una consecuencia inmediata del Corolario A.2.7 y de los lemas A.2.2 y A.2.3. □

Ejemplo A.2.1. Sean p y q dos primos impares distintos. Determinemos el número de soluciones de la ecuación $x^2 \equiv 1 \pmod{pq}$ y cuáles son éstas. Las soluciones de las ecuaciones $x^2 \equiv 1 \pmod{p}$ y $x^2 \equiv 1 \pmod{q}$ son, respectivamente, $x \equiv \pm 1 \pmod{p}$ y $x \equiv \pm 1 \pmod{q}$. Por tanto, $x^2 \equiv 1 \pmod{pq}$ tiene cuatro soluciones, dos de las cuales son $x \equiv \pm 1 \pmod{pq}$. Para encontrar las dos restantes habremos de solucionar el sistema:

$$\begin{cases} x \equiv 1 \pmod{p} \\ x \equiv -1 \pmod{q} \end{cases} \tag{A.6}$$

Para el sistema A.6, cualquier solución x deber cumplir $x \equiv 1 + pk$, para cierto entero k; pero además cumplirá $1 + pk \equiv -1 \pmod{q}$, es decir, $pk \equiv -2 \pmod{q}$. Tomemos

ahora enteros u y v tales que $1 = up + vq$; multiplicando la última congruencia por u resulta $k \equiv -2u \pmod{q}$, es decir, que existe un entero k' tal que $k = -2u + k'$ de donde resulta que $x = 1 - 2up + pqk'$. Por tanto, las dos soluciones restantes son $x \equiv \pm(1-2up) \pmod{pq}$. Por ejemplo, en el caso $p = 3$ y $q = 5$, como $1 = 3\cdot 2 + 5(-1)$, las soluciones son $x \equiv \pm 1 \pmod{15}$ y

$$\begin{aligned} x &\equiv \pm(1 - 2 \cdot 2 \cdot 3) \pmod{15} \\ &\equiv \pm(-11) \pmod{15} \\ &\equiv \mp 11 \pmod{15} \end{aligned}$$

es decir, la soluciones son: 1, 4 (por -11), 11 y 14 (por -1).

Apéndice B

El Teorema de Euler

B.1. Los Resultados de Fermat y Euler

Teorema B.1.1 (Fermat). *Sea* p *un número primo. Entonces para todo entero* a, $a^p \equiv a \pmod{p}$. *En particular, si* $p \nmid a$ *entonces* $a^{p-1} \equiv 1 \pmod{p}$.

Definición B.1.1. Para todo número natural no nulo m, representamos por $\phi(m)$ el número de números naturales a tal que $1 \leq a \leq m$ que son primos relativos con m. A la función ϕ se le llama *función de Euler*.

Observación B.1.1. Es claro que $\phi(1) = 1 = \phi(2)$. En general se tiene que $\phi(p) = p-1$, siempre que p sea un número primo.

Teorema B.1.2. *Sean* $p, a, b, n \in \mathbb{N}^*$. *Entonces:*

1. *Si* p *es primo, entonces* $\phi(p) = p - 1$.
2. *Si* p *es primo, entonces* $\phi(p^n) = p^n - p^{n-1} = (p-1)p^{n-1}$.
3. *Si* $(a, b) = 1$ *y* $2 \leq ab$, *entonces* $\phi(ab) = \phi(a)\phi(b)$.

Teorema B.1.3 (Euler). *Si* a *y* n *son números enteros tales que* $(a, n) = 1$, *entonces* $a^{\phi(n)} \equiv 1 \pmod{n}$.

B.2. Función Lambda de Carmichael

Definición B.2.1. Dado cualquier número natural no nulo n, la *función lambda de Carmichael* de n, $\lambda(n)$, es por definición el menor natural no nulo k tal que:

$$a^k \equiv 1 \pmod{n} \tag{B.1}$$

para todo a tal que $(a, n) = 1$.

Observación B.2.1. Dado cualquier número natural no nulo n, $\lambda(n)$ siempre existe porque el conjunto de los números k que cumplen (B.1) es no vacío, como demuestra el hecho de que $\phi(n)$ sea uno de sus elementos (cfr. Teorema B.1.3); así pues, $\lambda(n) \leq \phi(n)$. De hecho, la relación va más allá ya que $\lambda(n) \mid \phi(n)$. En efecto, existen unos únicos numeros naturales q y r tales que $\phi(n) = \lambda(n)q + r$ y $0 \leq r < \lambda(n)$; es claro entonces que para todo a tal que $(a, n) = 1$ se cumple $a^r \equiv 1 \pmod{n}$ y por la definición de $\lambda(n)$ ello no es posible salvo que r sea igual a 0, luego $\lambda(n) \mid \phi(n)$.

Teorema B.2.1. *Para cualesquiera números primos* p, $p_1, \ldots, p_k$ *y naturales no nulos* α, $\alpha_1, \ldots, \alpha_k$ *se cumple:*

$$\lambda(p^\alpha) = \begin{cases} \phi(p^\alpha), & \text{si } \alpha \leq 2 \text{ ó } p \geq 3 \\ \frac{1}{2}\phi(p^\alpha), & \text{si } \alpha \geq 3 \text{ y } p = 2 \end{cases}$$

$$\lambda(p_1^{\alpha_1} \cdots p_k^{\alpha_k}) = [\lambda(p_1^{\alpha_1}), \ldots, \lambda(p_k^{\alpha_k})]$$

Demostración. Sea p un primo impar. Un elemento de orden $\phi(p^\alpha)$ en $\mathbb{Z}^*_{p^\alpha}$ es denominado una *raíz primitiva*. Hay una raíz primitiva g módulo n cuando n es potencia de un primo impar. Como la menor potencia entera positiva de g que es congruente con 1 es g^{p^α}, ello demuestra que $\lambda(p^\alpha) \geq \phi(p^\alpha)$. Como sabemos que $\lambda(p^\alpha) \leq \phi(p^\alpha)$ (cfr. Observación B.2.1), entonces $\lambda(p^\alpha) = \phi(p^\alpha)$.

Cuando $p = 2$, sabemos que $\lambda(p^\alpha) \mid 2^{\alpha-2}$ para $\alpha \geq 3$. Una sencilla inducción demuestra que:

$$5^{2^{\alpha-3}} \equiv 1 + 2^{\alpha-1} \pmod{2^\alpha}$$

Así que el orden de 5 no divide a $2^{\alpha-3}$ pero es una potencia de 2, así que debe ser $2^{\alpha-2}$. Esto demuestra que $\lambda(2^\alpha) = 2^{\alpha-2} = \frac{1}{2}\phi(2^\alpha)$.

La última afirmación es una sencilla aplicación del *Teorema Chino del Resto.*. En particular, si $n = \prod_{i=1}^k p_i^{\alpha_i}$ entonces, para cualquier elección g_i de raíz primitiva módulo $p_i^{\alpha_i}$ (y $g_i = 5$ cuando $p_i^{\alpha_i}$ es una potencia de 2 mayor que 2), hay un único elemento g módulo n que es congruente con cada g_i módulo $p_i^{\alpha_i}$. Es fácil demostrar que el orden de g es igual al mínimo común múltiplo de los órdenes de los $\lambda(p_i^{\alpha_i})$. Por otra parte, un argumento similar basado en el Teorema Chino del Resto demuestra que cualquier elemento que supere ese mínimo común múltiplo debe ser 1 módulo n, puesto que es 1 módulo cada uno de los $p_i^{\alpha_i}$. □

Corolario B.2.2. *Para cualesquiera números natruales no nulos* m *y* n*:*

1. *Si* $m \mid n$, *entonces* $\lambda(m) \mid \lambda(n)$

2. $\lambda([m,n]) = [\lambda(m), \lambda(n)]$

Teorema B.2.3. *Sean* a *y* n *enteros positivos tales que* $(a,n) = 1$*. Entonces:*

$$a^{\lambda(n)} \equiv 1 \pmod{n}$$

B.3. El Teorema de Lagrange

Teorema B.3.1. *Sea* p *un número primo y sea* $f(x)$ *un polinomio de grado* $n \geq 1$ *tal que no todos sus coeficientes son múltiplos de* p*. Entonces* $f(x) \equiv 0 \pmod{p}$ *tiene a lo sumo* n *soluciones en un sistema completo de residuos módulo* p*.*

Corolario B.3.2. *Sea* p *un número primo y* d *un divisor de* $p-1$*; entonces la congruencia* $x^d - 1 \equiv 0 \pmod{p}$ *tiene exactamente* d *soluciones.*

Índice alfabético

Bibliografía

[1] Bauer, C.P. Secret History. The Story of Cryptology. Chapman & Hall/CRC, 2013.

[2] Bauer, F.L. Decrypted Secrets. Methods and Maxims of Cryptology. Springer-Verlag, fourth edition, 2007.

[3] Biham, E. and Shamir, A. Differential Cryptoanalysis of the Data Encryption Standard. Springer-Verlag, 1993.

[4] Blakley, G.R. and Borosh, I. Rivest-shamir-adleman public-key cryp. do not always conceal messages. Comp. and Maths. with Appls., 5:169–178, 1979.

[5] Delfs, H. and Knebl, H. Introduction to Cryptography. Springer-Verlag, third edition, 2015.

[6] Helen Fouché Gaines. Cryptanalysis. A study of ciphers and their solution. Dover Publications Inc., 1956.

[7] Hoffstein, J., Pipher, J., and Silverman, J.H. An Introduction to Mathematical Cryptography. Springer-Verlag, 2014.

[8] Koblitz, Neal. A Course in Number Theory and Criptography. Springer-Verlag, 1994.

[9] Konheim, Alan G. Cryptography: A Primer. John Wiley & Sons, 1981.

[10] Konheim, Alan G. Computer Security and Cryptography. John Wiley & Sons, 2007.

[11] Kumanduri, R. and Romero, C. Number Theory with Computer Applications. Prentice-Hall, 1998.

[12] López Guerrero, M.A. Ortega Triguero, J.J. and García del Castillo Crespo, E.C. Introducción a la Criptografía. Historia y Actualidad. Ediciones de la Universidad de Castilla-La Mancha, 2006.

[13] Sarasa López, M.A. Pastor Franco, J. and Salzar Riaño, J.L. Criptografía Digital. Fundamentos y Aplicaciones. Prensa Universitaria de Zaragoza, 2001.

[14] Rivest, Ron. Remarks on a proposed cryp. attack on the mit public-key cryp. Cryptologia, 2:62–65, 1978.

[15] Salomaa, A. Public-Key Cryptography. Springer-Verlag, 1990.

[16] Salomon, D. Foundations of Computer Security. Springer-Verlag, 2006.

[17] Schneier, Bruce. Applied Cryptography. John Wiley & Sons, 1994.

[18] Stinson, D.R. Cryptography. Theory and Practice. Chapman & Hall/CRC, third edition, 2006.

[19] Wiener, M.J. Cryptanalysis of short rsa secret exponents. In EUROCRYPT-89, 1991.