"ESTUDIO MONOGRÁFICO SOBRE TÉCNICAS DE CRIPTOGRAFÍA"

TRABAJO DE GRADUACIÓN

PREPARADO PARA LA FACULTAD DE INGENIERÍA

PARA OPTAR AL GRADO DE:

INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN

POR:

NURY MERCEDES MARTÍNEZ SILVA

JUNIO DE 2004

SOYAPANGO – EL SALVADOR – AMÉRICA CENTRAL

UNIVERSIDAD DON BOSCO

RECTOR

ING. FEDERICO MIGUEL HUGUET RIVERA

SECRETARIO GENERAL

LIC. MARIO RAFAEL OLMOS

DECANO DE LA FACULTAD DE INGENERÍA

ING. ERNESTO GODOFREDO GIRÓN

ASESOR DEL TRABAJO DE GRADUACIÓN

ING. OSCAR DURÁN VIZCARRA

JURADO EVALUADOR

LIC. SANTIAGO ABARCA

ING. EDUARDO RIVERA

ING. CARLOS GIOVANNI VÁSQUEZ

TUTOR DEL TRABAJO DE GRADUACIÓN

LIC. JORGE MAURICIO COTO

UNIVERSIDAD DON BOSCO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN

JURADO EVALUADOR DEL TRABAJO DE GRADUACIÓN

LIC. SANTIAGO ABARCA	ING. EDUARDO RIVERA
JURADO	JURADO

ING. CARLOS GIOVANNI VÁSQUEZ

JURADO

ING. OSCAR DURÁN VIZCARRA	LIC. JORGE MAURICIO COTO
ASESOR	TUTOR

AGRADECIMIENTOS

Gracias a Dios y a la Virgen María por haberme dado fuerzas para poder finalizar este trabajo, uno de mis tantos sueños, y a las personas que a lo largo de mi camino, de una u otra forma, contribuyeron a este logro:

A mi madre,

Quien estoy segura siempre ha seguido mis trabajos, esfuerzos, dificultades y progresos.

A mis hermanos,

Por ser ejemplo de superación y porque esos recuerdos de infancia, y esos niños que aún queremos conservar dentro de nosotros, nos permiten reunirnos y creer que el tiempo no ha pasado.

A la familia Silva,

Mi abuelita Herminia, y mis tías Yolanda y Elsa, por su cariño, comprensión y por su permanente apoyo.

A la familia Acuña,

Mis tíos Carlos, Arnoldo, Agenor y mi primo Carlos, por brindarme siempre ese cariño tan especial y sus palabras de aliento. A mi tía Carmen por esa dulzura y sencillez con las que siempre me animó a superarme.

A Oscar Durán y Margot de Durán,

Por haberme permitido invadir su casa mientras trabajaba en este proyecto, por brindarme su confianza y cariño e incentivarme a seguir adelante.

A Oscar Durán Vizcarra,

Por creer en mí y en el proyecto, por exigirme y enseñarme a trabajar con calidad, por ser un gran amigo, por su apoyo y ayuda incondicional.

A mis amigos y compañeros de trabajo,

Cecyl, Ernesto, Luis y Nixon, por esa amistad tan bonita que nos une y porque desinteresadamente realizaron mis tareas mientras yo me encontraba dedicada a finalizar este proyecto. A mi jefe, Carlos Gómez, por todo el tiempo que me permitió tomarme para dedicarme a concluir esta ardua tarea.

A la familia Thumann,

Por sus palabras de aliento, cariño, colaboración y apoyo.

A mi amiga Carmen Aida Mojica,

Quien tanto me apoyó para concretar esta meta y quien ahora desde el cielo se regocija al verla finalizada.

DEDICATORIA

A mi padre, Francisco Javier Martínez,

Por haberme enseñado a luchar, a vivir la vida con optimismo, y a creer en mí y en mis propias fuerzas.

Por haberme corregido a tiempo para luego dejarme tomar mis propias decisiones, y sentirse orgulloso de ellas.

Por nunca haber escatimado esfuerzos para instruirme.

Por contagiarme de ese espíritu de superación y esas ganas infinitas de vivir que él mismo ha mostrado.

Por enseñarme el significado de la palabra sacrificio.

Por señalar mis errores con el único afán de hacerme crecer como persona.

Por enseñarme que aun en la peor de las situaciones siempre la vida deja una lección y que por lo tanto la palabra derrota no existe.

Por el cariño, la comprensión, el respeto y confianza que siempre ha tenido hacia mí.

Porque un día el soñó con que su única hija se graduara de la Universidad y porque dedicó su vida a guiarme por ese camino.

Porque me dio lo que nunca tuvo.

Papá, este triunfo es, sobre todo, para tí.

INDICE

ANEXOS

ANEXO I – Criptoanálisis
ANEXO II – Ejemplo de la primera iteración de un texto al que se aplica el algoritmo DES
ANEXO III – Tabla comparativa entre cifrado de llave secreta y cifrado de llave pública
ANEXO IV – Tabla comparativa de estructuras de algunos algoritmos de cifrado de bloques
ANEXO V – Tabla comparativa del rendimiento de algunos algoritmos de cifrado de bloque
ANEXO VI – El Modelo OSI
ANEXO VII – SSL - Secure Socket Layer
ANEXO VIII – Kriptonita - Aplicación para cifrado y descifrado de datos

Introducción

Esta monografía recoge información sobre las bases, historia y una importante serie de algoritmos utilizados para el cifrado de datos, con el propósito de que el lector cuente con información suficiente que le permita explorar desde distintas perspectivas el campo de la criptografía.

A fin de lograr lo anterior, este trabajo ha sido estructurado en cinco capítulos. En el primero se exploran aspectos generales que están a la base de este campo de estudios; se hace particular énfasis en la teoría de la información, debido al impulso que esta brindó al desarrollo de la criptografía y el criptoanálisis contemporáneos.

En el segundo capítulo se presenta una reseña histórica de la evolución que la criptografía ha experimentado desde sus orígenes hasta la actualidad. Se destacan tres períodos a lo largo de la historia que están delimitados por determinantes cambios de paradigmas.

El tercer capítulo presenta información sobre algunas técnicas importantes de las que se hace uso en algunos de los algoritmos de cifrado contemporáneos. Se da especial énfasis a los modos de operación más utilizados y se efectúa una comparación entre ellos.

El cuarto capítulo esta destinado a la descripción de una serie de algoritmos de llave privada. Se brinda mayor atención a los algoritmos DES y Rijndael por la importancia que reviste el hecho de que se les haya adoptado por estándares.

El quinto capítulo aborda el estudio de los algoritmos de llave pública. Adicionalmente se ofrecen detalles acera de sistemas híbridos en los que se explotan las cualidades de los algoritmos de llave

privada y de llave pública, también se provee información sobre firmas digitales y funciones resumen.

El lector encontrará también una serie de anexos que pretenden documentar de forma resumida aspectos importantes que permiten complementar las ideas vertidas en este trabajo. Además, se incluye el código fuente de la aplicación que se desarrolló para poder realizar pruebas con tres de los algoritmos descritos a lo largo del documento.

La elaboración del trabajo requirió una intensa consulta de material bibliográfico. Es de destacar que mucha información importante esta contenida en artículos y que por lo tanto la información disponible se encuentra dispersa. Se ha procurado recoger en este trabajo la información que se consideró más relevante integrándola en este documento.

Objetivos

Objetivo General

Estudiar, describir y analizar el funcionamiento de las diferentes técnicas que se utilizan para el cifrado de datos.

Objetivos específicos

Investigar las técnicas de cifrado existentes, sus orígenes y el porqué son necesarias.

Generar un documento que describa el funcionamiento de cada una de las técnicas de encriptación.

Comparar las técnicas sometidas a análisis considerando aspectos como eficiencia y seguridad.

Desarrollar una aplicación utilizando lenguaje C, que sirva como herramienta de apoyo en la enseñanza de materias relacionadas a la criptografía de datos.

Alcances

El estudio describe técnicamente los orígenes de la criptografía.

Se identifican las técnicas de criptografía existentes y los diferentes campos en que éstas son aplicadas.

Se describe paso a paso el funcionamiento de las técnicas de criptografía existentes y sus áreas de aplicación.

Se demuestra el proceso de cifrado de datos para algoritmos mediante una aplicación desarrollada para tal fin.

Limitaciones

El estudio no involucra el desarrollo de nuevas técnicas para el cifrado de datos.

La aplicación demostrativa sólo incluye algunos de los algoritmos más importantes en la actualidad.

1

Aspectos introductorios

La necesitad de privacidad en la información es tan antigua como la comunicación. En la actualidad esta necesidad adquiere mayor relevancia debido al nivel de desarrollo y expansión de las de redes de comunicación, así como al extensivo uso de equipo informático en el procesamiento y transmisión de información crítica.

El proceso de comunicación puede ser modelado de acuerdo a lo que se presenta en la figura 1.1

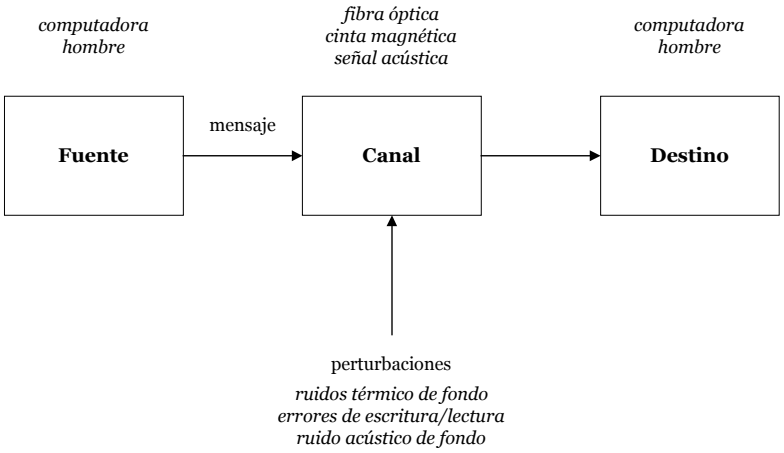


Figura 1.1 Modelo genérico de sistema de comunicación.

La Figura 1.1 representa el esquema de comunicación comúnmente conocido como el paradigma de Shannon. Una fuente genera un mensaje con la intención de hacerlo llegar a un destino. La fuente y el destino son dos entidades separadas (eventualmente distantes) que están vinculadas por un canal, que es el soporte de la comunicación por una parte, pero que por otra es donde se alojan perturbaciones. Las perturbaciones tienen por efecto crear una diferencia entre el mensaje emitido y el recibido. Estas perturbaciones son de naturaleza aleatoria, es decir que no es posible (ni para la fuente ni para el destino) prever de forma certera su efecto. El canal no sólo está expuesto al ruido, sino también a ataques.

Existen diferentes tipos de ataques, tal como se ilustra en la Figura 1.2, el ataque puede ser pasivo, si se limita a la intercepción del mensaje o activo cuando el mensaje ha sido capturado, modificado o eliminado.

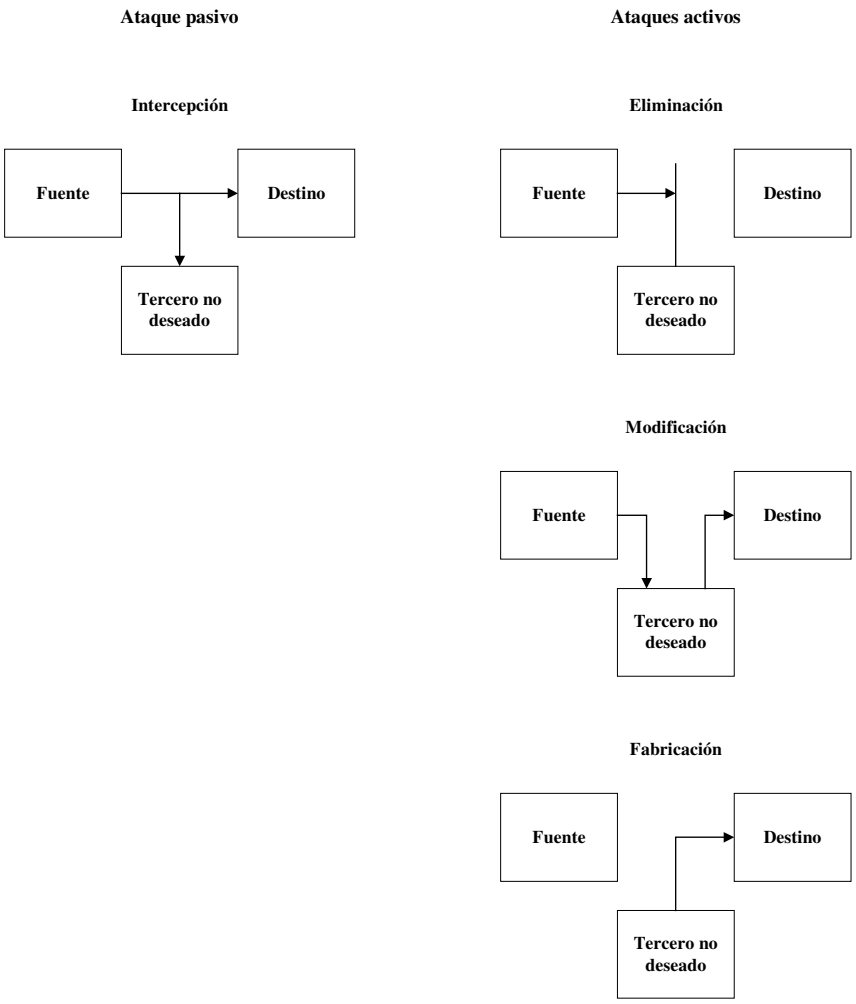


Figura 1.2 Tipos de ataque.

En trabajos realizados en Bell Labs por Claude Shannon[1], se contempla la conveniencia de efectuar codificación de fuente y canal para contrarrestar los efectos nocivos de las perturbaciones en la comunicación.

Los resultados fundamentales de la teoría de la información fueron establecidos públicamente por Shannon desde 1948 (a la óptica de las telecomunicaciones), y en particular el hecho capital, y totalmente imprevisto en la época, de que es posible establecer una transmisión de información exenta de errores, pese a la existencia de ruido de fondo, pero ello supone una representación apropiada de la información (normalmente identificada con en término *codificación*) y la imposición de restricciones al caudal (o *densidad*) de información transmitida, que dependen de las características del canal. Sin embargo, hubo que esperar los desarrollos recientes en informática y telecomunicaciones para ver aparecer sistemas que se aproximen efectivamente a las condiciones extremas enunciadas por Shannon.

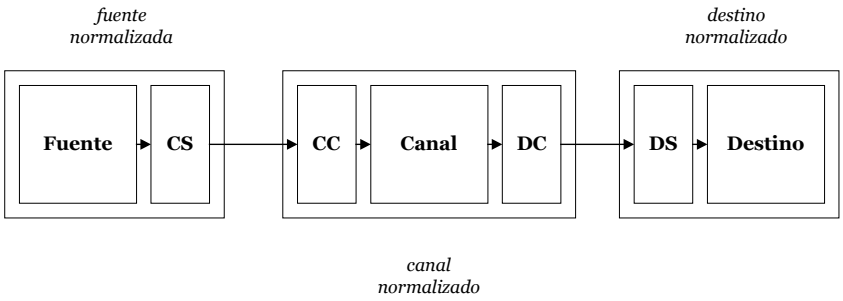


Figura 1.3 Modelo de comunicación con codificación/decodificación de fuente/canal.

El modelo de comunicación "ideal" puede ser esquematizado tal como se representa en la Figura 1.3. Ahí se ven aparecer dos conjuntos de codificación CS y CC (y los elementos de decodificación correspondientes DS y DC): CS codifica el mensaje emitido por la fuente de manera que se elimine (o reduzca) la redundancia que contenga; CC codifica los mensajes a la entrada del canal, introduciendo redundancia bajo una forma apropiada que permita la utilización del canal sin errores (o con una tasa de errores que responda a las especificaciones técnicas del sistema). Es de notar que en este esquema las conversiones físicas necesarias para la utilización del canal están implícitamente representadas en el bloque "canal". Nótese que el estudio de canales físicos y técnicas de conversión asociadas constituye una parte importante de las telecomunicaciones, de la que no nos ocupamos sin embargo en el marco de este trabajo.[2]

Si bien los trabajos de Shannon datan de los años 40, a lo largo de la historia siempre se ha recurrido a técnicas que procuran proteger la información del conocimiento de personas distintas del destinatario originalmente previsto por quien la emite. Estas técnicas pueden consistir en ocultar la información, en representarla mediante códigos de uso restringido o en el desordenamiento de los elementos del mensaje. El conjunto de estas técnicas recibe el nombre de Criptografía.[3]

---

[1] Shannon, C. *"The Mathematical Theory of Communication"*. Reprinted with corrections from The Bell System Technical Journal. Vol. 27. pp. 379-423, 623-656. July-October, 1948.

[2] Wehenkel, L. Théorie de l'information et du codage. Faculté des Sciences Appliquées. Université de Liège. Liège. 2001. pp. 7-8.

[3] Este término procede del las palabras griegas *"kryptos"* que significa oculto o secreto, y *"graphein"* que significa escritura o tratado

También existe una serie de técnicas cuyo objeto es volver ininteligible a terceros la información que fluye entre fuente y destino sin el consentimiento de éstos. El conjunto de tales técnicas se denomina Criptoanálisis. La suma de conocimientos de la criptografía y el criptoanálisis integra la Criptología.

Para volver ininteligible un mensaje, al que se le llama *mensaje claro*, la fuente o emisor aplicará un método criptográfico al que de forma genérica se denomina *algoritmo,* así como una *llave,* que especifica los detalles exactos para un procedimiento criptográfico en particular, esto arroja al canal de comunicación la misma información contenida en el mensaje claro pero de forma que no es comprensiva para terceros, a esto se le llama *criptograma.* Para recuperar el mensaje claro, que era la forma que tenía originalmente la información, el destino o receptor debe tomar el criptograma y aplicarle el algoritmo y la llave previamente convenidos con la fuente o emisor. Todo esto se ilustra en la figura 1.4.

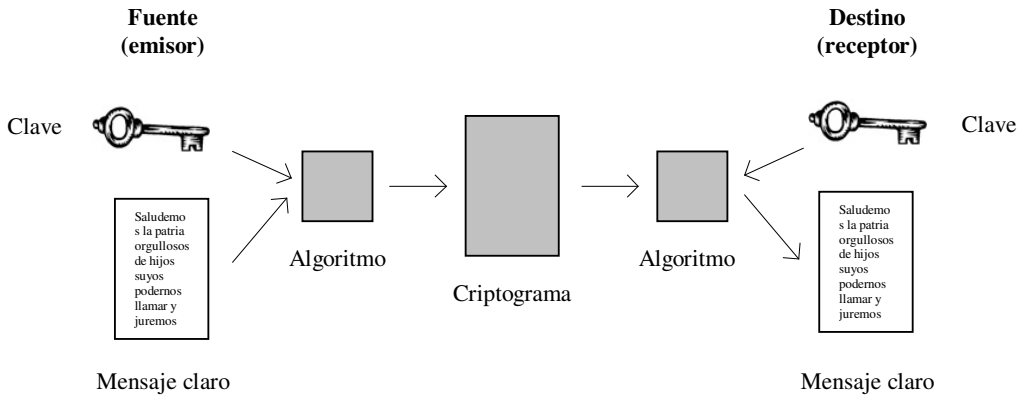


Figura 1.4 Esquema de un proceso criptográfico

En la jerga técnica se suele llamar *encriptamiento* o *crifrado* al proceso por medio del cual se vuelve ininteligible un mensaje claro, y se habla casi indistintamente de *decriptamiento* o *descifrado* para referirse al proceso contrario (recuperar el mensaje claro a partir del criptograma recibido). A este respecto es conveniente precisar que en español los términos *encriptamiento* y *decriptamiento* no existen. En inglés y en francés, sin embargo, existe cierto matiz entre las nociones de *descifrado* y *decriptamiento*: mientras el término *descifrado* (*to decipher* en inglés, y *dechiffrer* en francés) se refiere a la recuperación del mensaje claro por parte del destinatario a partir de la aplicación del algoritmo y la llave convenidos con la fuente, el término *decriptar* (*to decrypt* en inglés, y *decrypter* en francés) se refiere a obtener el mensaje claro sin conocimiento a priori del mecanismo criptográfico utilizado, por lo cual este último término, siendo más juiciosos en el uso del lenguaje, debería utilizarse más para aludir al caso de la intervención de un tercero que trata de conocer el mensaje claro sin el aval de la fuente y el destino, es decir, en caso de un ataque.[4]

[4] Real Academia Española. Diccionario de la lengua española. Real Academia Española. Madrid. 1992. 21ª Edición.
CLE International. Dictionaire du Français. Dictionaires LeRobert. CLE International. Paris. 1999
Stern, J. La science du secret. Éditions Odile Jacob. Paris. 1998. pp. 10-11.

1.1 Ramas de la Criptografía

Antes se dijo que para proteger la información, a lo largo de la historia, se ha recurrido a tres tipos de técnicas:

- Aquellas en las que la información se oculta reciben el nombre de esteganografía;
- Aquellas en las que la información es representada mediante códigos que sólo conocen la fuente y el destino se denominan codificación; y
- Aquellas en las que el mensaje es desordenado mediante la sustitución o cambio de posición de sus elementos, recibe el nombre de cifrado.

Lo anterior se ilustra en el esquema de la figura siguiente:

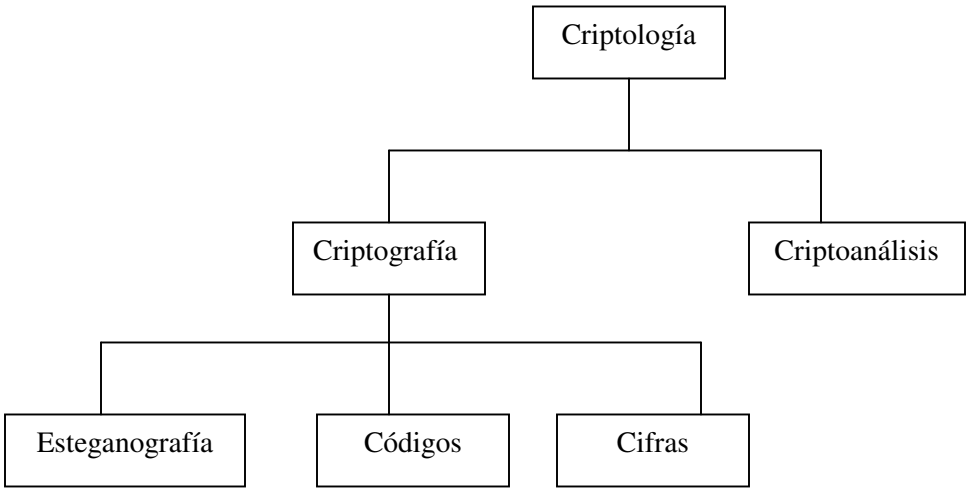


Figura 1.5 Técnicas comprendidas en la Criptología.

En los siguientes apartados nos dedicaremos a la descripción de las tres ramas de la criptografía.

1.1.1 Esteganografía

Esta rama de la criptografía consiste camuflar o enmascarar el mensaje de forma que mientras se le transporta permanezca oculto. Es decir que para proteger la información, pretende esconder la existencia del mensaje.

El término *esteganografía*, que viene del griego *stegos* (cubierta), significa "escritura oculta" o "escritura encubierta" y es el conjunto de técnicas que nos permiten ocultar o camuflar cualquier tipo de datos.

La esteganografía tiene un origen muy antiguo. Una de las primeras y sencillas técnicas esteganográficas era la tinta invisible que consistía en ocultar los mensajes escritos con jugo de limón en un papel en el que al calentarlo aparece lo escrito.

Herodoto narra en sus Historias los conflictos entre Grecia y Persia en el siglo V AC. De acuerdo a lo que relata, la utilización de ciertas técnicas criptográficas salvo a Grecia de ser conquistada por Jerjes, rey déspota de los persas.

Es precisamente Herodoto el primero en registrar el uso de técnicas esteganográficas. Un cierto Histio, queriendo establecer contacto secreto con su superior, Aristágoras de Mileto, escogió a un esclavo leal, hizo afeitar su cabeza y escribió sobre su cuero cabelludo el mensaje que deseaba enviar; esperó a que le creciera de nuevo el cabello y lo envió al encuentro de Aristágoras con la instrucción de hacerse rapar nuevamente para dejar visible el mensaje.[5]

Herodoto también relata que para informar a los espartanos de un ataque inminente de los persas, el rey Demaratos utilizó una estrategia muy elegante: pegó tablillas, les retiró la cera, grabó en la madera el mensaje secreto y los recubrió nuevamente con cera. De este modo, las tablillas, aparentemente vírgenes, no llamaron la atención. El problema era que los griegos no sabían qué ocurría cuando Gorgo, la esposa de Leonidas, tuvo la idea de raspar la cera.[6].

Las técnicas antes explicadas se han adaptado a los nuevos tiempos y, en muchas ocasiones, aparecen combinadas con métodos criptográficos. Las técnicas esteganográficas más comunes en informática se basan en ocultar la información en archivos gráficos[7] o archivos de sonido.

Cuando se usan archivos de sonido la información oculta aparece como ruido de fondo, pudiendo confundirse fácilmente con una simple grabación con algo de ruido.

Para ocultar información dentro de una imagen requiere de dos archivos. El primero es la imagen que contendrá la información a ser ocultada, llamada imagen cubierta, y el segundo archivo es el mensaje a ocultar, este mensaje puede ser texto claro, cifrado u otras imágenes. Luego que ambos archivos son combinados, la imagen resultante es denominada estego-imagen[8].

1.2 Codificación

Un código es esencialmente un lenguaje secreto inventado para esconder el sentido de un mensaje.

---

[5] Herodoto. Los nueve libros de la Historia. Clásicos Jackson. México DF. 1966. pp. 291-294.
[6] Herodoto. Op. Cit. pp. 348-356.
[7] Para una computadora, una imagen es un arreglo de números que representan intensidades de luz en varios puntos (píxeles). Un tamaño común de imagen es 640 X 480 píxeles y 256 colores, u 8 bits por píxel, así, una imagen podría contener alrededor de 300kb de datos. Las imágenes digitales son típicamente almacenadas en archivos de 24 u 8 bits. Una imagen de 24 bits provee más espacio para ocultar información; sin embargo, puede ser una imagen muy grande (a excepción de imágenes JPEG). Todos las variaciones de color por píxeles son derivadas de los tres colores primarios: rojo, verde y azul. Cada color primario es representado por 1 byte; una imagen de 24 bits ocupará 3 bytes por píxel para representar un color determinado, estos tres bytes pueden ser representados como valores hexadecimales, decimales y binarios. En muchas páginas web, el color de fondo es representado por un número hexadecimal, un fondo color blanco tendrá el valor de FFFFFF: 100 por cierto rojo (FF), 100 por ciento verde (FF) y 100 por ciento azul (FF). Su valor decimal es 255, 255, 255 y su valor binario 11111111, 11111111, 11111111, los cuales son los tres bytes que componen el color blanco.
[8] Johnson, N., Jajodia, S. *Exploring Steganography: Seeing the Unseen*. Computer Practices. S/l,. S/f. Pp. 26-34.

Sea el caso del texto “Águila 3, aquí águila 2, el león está en la jaula”. En esta frase, por ejemplo, el agente “Águila 2” usa palabras código para avisar al agente “águila 3” que el sujeto sometido a espionaje (león) entró en una casa (jaula).

Los códigos de espionaje o militares son denominados comúnmente de *números-código* en lugar de *palabras-código*. Se utilizan libros de código que proveen un diccionario de *números-código* y sus respectivas palabras.

El mensaje anterior podría ser codificado como: 85772 24799 10090 59980, donde, “85772" significa “águila 3”, "24799" significa “águila 2”, "10090" significa “león” y "59980" significa “jaula”. El conjunto de *palabras–código* y *números-código* es denominado grupos de código (59980 - jaula). Las palabras que representan son llamadas de texto claro (casa, residencia).

Originalmente, los grupos de código estaban en el mismo orden que sus respectivas palabras claras. Por ejemplo, una palabra que comenzaba en “a” tendría un número de código bajo y una palabra que comenzara por “z” tendría uno alto. Esto permitía que el mismo libro de código fuera utilizado tanto para codificar como para decodificar un mensaje.

Estos códigos eran relativamente previsibles y permitían que lectores indeseados descubriesen el patrón y por consiguiente el mensaje, revelando secretos. Para dificultar más las cosas a posibles intrusos, los creadores de códigos elaboraron unos donde la relación entre los grupos y las palabras no es previsible.

Esto significa que se necesitan dos libros de códigos, uno para localizar el texto claro y encontrar los grupos de código correspondientes (codificar) y otro para localizar los grupos de código y texto claro correspondiente (decodificar). Estos códigos de dos partes son más difíciles de crear y usar, pero tienen la ventaja de ser más difíciles de romper.

1.2.1 Ruptura de códigos

Descifrar un mensaje codificado es un tanto parecido a la traducción de un documento a otro idioma, donde la tarea básica es construir un “diccionario” de los grupos de código con las palabras claras que ellos representan.

Una de las características de un código simple es el hecho de que algunas palabras son más frecuentes que otras, como “de” y “un” en español. En mensajes telegráficos el grupo de código para “punto” (fin de una oración) generalmente también es común. Esto ayuda a definir la estructura de un mensaje en términos de oraciones, y hasta de su sentido.

Lo anterior también ayuda a romper un código y juntar muchos mensajes codificados con el mismo código y después obtener evidencias como el lugar de donde fueron enviadas y hacia donde las dirigieron; la hora en que fue enviado; eventos que ocurrieron antes y después que el mensaje fuera enviado y los hábitos de las personas que los enviaron.

Se pueden usar varios trucos para insertar información deliberadamente. Por ejemplo, hacer un ataque contra el enemigo en una determinada hora en un determinado lugar y después examinar mensajes codificados enviados por ellos como reacción al ataque. Los errores de codificación son especialmente útiles y es claro que tarde o temprano las personas cometerán errores, algunos de ellos desastrosos.

La forma más obvia de romper un código es obteniendo el libro de códigos mediante un soborno, robo o invasión. Esta es la flaqueza de los códigos. Un buen código puede ser más difícil de romper que un cifrado de sustitución, pero la elaboración y distribución de libros de código pueden ser deficientes.

Preparar un nuevo código es como construir un nuevo idioma y elaborar un diccionario para ella es un trabajo intenso, si un código estuviera comprometido, toda tarea requeriría ser replanteada, lo que significaría mucho trabajo para los codificadores y los usuarios del código. En la práctica, después de un tiempo de uso los códigos son alterados para frustrar los intentos de ruptura. Automáticamente esto trae consigo mucho tiempo y trabajo.

Una vez habiendo sido creados, la distribución de los códigos es complicada y fácilmente puede comprometerlos. Se dice que dos personas pueden guardar un secreto sí una de ellas estuviera muerta. Puede ser exagerado, pero el hecho es que un secreto es más difícil de ser mantenido mientras más personas tengan conocimiento de él. Sí sólo una pocas personas hicieran uso de un código, éste estaría razonablemente seguro, pero si ejércitos enteros hacen uso del mismo código, la dificultad de mantenerlo seguro es mucho mayor.

1.3 Cifrado

Para el análisis de las cifras es interesante clasificarlas en grupos basados en su funcionalidad. Es a este tipo de procedimiento criptográfico al que se dedicará mayor atención a lo largo de éste documento, dado que es el que sustenta la mayoría de aplicaciones criptográficas de la actualidad.

1.3.1 Sustitución y transposición

El cifrado por sustitución es un criptograma en el cual las letras originales del texto original, tratadas individualmente o en grupos de longitud constante, son sustituidas por otras letras, figuras, símbolos o una combinación de éstos de acuerdo a un sistema definido y una llave.

El cifrado por transposición es un criptograma en el cual las letras originales sólo son reordenadas de acuerdo a un sistema definido.

En otras palabras, para cifrar se recurre a sustitución o transposición, la diferencia fundamental entre esos métodos es que en la sustitución el valor normal o convencional de las letras del texto original se cambia, sin que su posición sea modificada; en la transposición sólo se altera la posición de las letras del texto original sin que haya modificaciones en su valor normal o convencional.

Como los métodos de cifrado son radicalmente diferentes, los principios involucrados en el criptoanálisis de ellos también son fundamentalmente diferentes.

El cifrado puede efectuarse por sustitución o transposición. La sustitución puede ser monoalfabética o polialfabética.

El método monoalfabético puede ser de tres tipos: monográmico, poligrámico y tomográmico. A continuación se describen las características de cada uno:

1.3.1.1 Sustitución monoalfabética monográmica

En la sustitución monoalfabética, también conocida como sustitución simple, se reemplaza uno de cada dos caracteres del texto original por otros, de acuerdo con una tabla preestablecida para obtener el texto cifrado. Como consecuencia, la frecuencia de aparición de las letras (números o símbolos) del mensaje cifrado es la misma de las letras del idioma usado en el mensaje original.

Se le llama monográmica (o monográfica) porque cada letra del mensaje original es sustituida por una sola letra número o símbolo. Por tanto, la longitud del mensaje cifrado es la misma que la del mensaje original.

Las cifras más antiguas de sustitución monoalfabéticas son el Atbash y el código de César. En la criptografía contemporánea, se sustituyen bloques de bits en lugar de caracteres. El principio, sin embargo, es el mismo.

Este tipo de cifra solo es relativamente seguro sólo en textos muy cortos. Un simple criptoanálisis probabilística, basado en la caracterización estadística del idioma, es suficiente para descifrar el texto. Para aumentar la seguridad de estas cifra se pueden usar nulos, homófonos, polifónicos o repertorios (o nomenclaturas).

1.3.1.2 Sustitución monoalfabética poligrámica

El término poligrámica o poligráfica denota la utilización de varios caracteres. Esta técnica de sustitución tiene las mismas características de la sustitución simple, con la diferencia de que se sustituye uno o más caracteres del mensaje original por uno o más letras, números o símbolos. Por tanto, la longitud del mensaje cifrado no siempre es la misma que la del mensaje original. Esta es la sustitución más genérica posible.

Dentro de las sustituciones monoalfabéticas poligrámicas figura la llamada homofónica. El término homofónico procede del griego y significa “mismo sonido”. Es el concepto de tener tres secuencias diferentes de letras que son pronunciadas de forma semejante. En criptología es un cifrado que traduce un único símbolo del texto claro para uno de muchos símbolos cifrados, todos con el mismo significado.

1.3.1.3 Sustitución monoalfabética tomográmica

Los sistemas tomográmicos son aquellos en los cuales cada letra es representada por un grupo de dos o más letras o números. Estas letras o números son obtenidos mediante un cifrado por sustitución o por transposición separada.

Se puede decir que la sustitución monoalfabética monográmica es una sustitución uniliteral (no confundir con unilateral), pues se cambia cada uno de los caracteres del texto claro por otro cifrado. Cuando los grupos de sustitución están constituidos por más de una letra o símbolo, decimos que la sustitución es multiliteral.

1.3.1.4 Sustitución polialfabética

El término alfabeto se aplica al conjunto de símbolos que serán utilizados para sustituir los símbolos (letras) originales. En una sustitución polialfabética se utilizan múltiples alfabetos para llevar a cabo la sustitución de un mismo mensaje.

Los alfabetos no requieren necesariamente tener orígenes diferentes, por ejemplo, un alfabeto romano y otro cirílico. El simple hecho de alterar el orden en la secuencia de las letras ya caracteriza un "nuevo" alfabeto. Por ejemplo, z-y-x-...-c-b-a es un alfabeto de sustitución; b-a-d-c-... es un alfabeto de sustitución diferente. Si ambos fueran utilizados para cifrar un mismo mensaje, reemplazando las letras originales, entonces se trataría de una sustitución polialfabética.

La forma más antigua de cifrado polialfabético fue desarrollada por Leon Battista Alberti en 1466. Su sistema consistía en escribir el texto cifrado en letras minúsculas y usar las mayúsculas como símbolos denominados indicadores, para enfatizar cuando la sustitución cambiaba. El alfabeto cifrante del disco de Alberti estaba ordenado e incluye los dígitos del 1 al 4, usados para formar palabras código de un pequeño vocabulario. Posteriormente, formas más modernas fueron desarrolladas, en las cuales la sustitución cambiaba para cada letra del texto claro.

1.3.1.5 Sustitución polialfabética con clave progresiva

Un sistema de clave progresiva es uno donde los alfabetos cifrantes (o claves) son usados unos después de otros en un orden normal. Este cifrado fue publicado póstumamente en un libro de Johannes Trithemius que apareció en 1518. En la tabla recta de Trithemius, la clave ABCD...Z es usada como alfabeto regular en la forma indicada por el autor.

1.3.1.6 Sustitución polialfabética con palabra clave

Es una sustitución polialfabética donde una palabra clave indica los alfabetos cifrantes que deben ser usados. A pesar de ser este sistema atribuido a Vigenère, debe originalmente su origen a Giovanni Battista Bellaso en 1553. Diez años más tarde, en 1563, Giambattista Della Porta agregó el uso de alfabetos mixtos a éste sistema.

1.3.1.7 Sustitución polialfabética con clave automática

En un sistema de este tipo hay una clave que indica la elección inicial del alfabeto cifrante y después el propio mensaje determina los alfabetos subsecuentes. La primera propuesta fue de Girolamo Cardano, sin embargo poseía fallas. Fue Blaise de Vigenère quien publicó la forma moderna del cifrado con clave automática en 1585.

2

Evolución histórica

La criptografía ha pasado, desde sus orígenes hasta la fecha, por tres etapas, que son la era artesanal o manual, la era técnica o mecánica y la era digital[9]. La transición de un período a otro se dio por una especie de ruptura o salto cualitativo que permitió a los criptólogos superar obstáculos que limitaban el florecimiento de su arte. En la Figura 2.1 se ilustran los límites de las eras antes mencionadas.

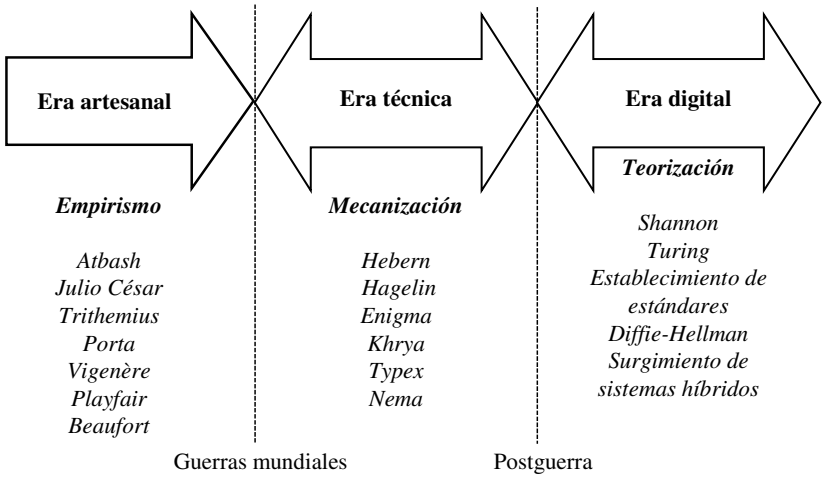


Figura 2.1 Eras de la criptografía

[9] Jacques Stern se refiera a esta última como "era de las paradojas", pero ubica su inicio no con los trabajos de Claude Shannon, sino unos treinta años más tarde con los de Diffie-Hellman. Stern J. Op. Cit. pp 9-17

2.1 Era Artesanal

La era artesanal se extiende desde los orígenes de la criptografía hasta el período comprendido entre las dos guerras mundiales, a pesar de que ciertos procedimientos continuaron coexistiendo con métodos mecánicos durante la segunda guerra mundial. Este período tiende a ejercer entre legos cierta fascinación, quizá porque la criptología de la Edad Media y el Renacimiento realzaba un *know how* esotérico que la relacionaba con la Alquimia[10] y la Cábala[11].

Desde que la confidencialidad fue una necesidad social, la criptografía se convirtió en un corolario interesante de la escritura; por consiguiente no son sorprendentes las primicias encontradas desde la época de los jeroglíficos egipcios o textos cuneiformes[12].

En 1900 A.C., un escriba egipcio usó un tipo de jeroglíficos en la inscripción de tumbas. Su intención no era hacer más difícil la lectura del texto, sino dar realce a lo escrito. La inscripción no contenía un mensaje secreto, pero incorporaba uno de los elementos esenciales de la criptografía: una transformación deliberada de la escritura[13].

Similarmente, varias tabletas conteniendo escritura cuneiforme transforman las fórmulas finales (llamadas *colofones*) de firma y fechado de las tabletas con la ayuda de signos raros, algunos incluso representan el nombre del escriba en serie cifrada[14].

En la antigua China se escribían mensajes sobre seda fina. Después se hacía una bolita que era envuelta en cera. En seguida, el mensajero engullía la bola. Evidentemente, se trataba de una forma de esteganografía. Se dice que la cultura china no desarrolló criptografía a partir de técnicas de cifrado debido a que, aunque el uso de la escritura era extremadamente antiguo, la literatura estaba siempre restringida a una minoría en la cual el mero acto de escribir algo equivalía a codificarlo[15].

El primer caso claro de uso de métodos criptográficos se dio durante la guerra entre Atenas y Esparta (431-404 AC), el cifrado se basaba en la alteración del mensaje original mediante la inclusión de símbolos innecesarios que desaparecían al enrollar la lista en un rodillo llamado

---

[10] El Diccionario de la Real Academia Española la define como el conjunto de especulaciones y experiencias generalmente de carácter esotérico, relativas a la transmutaciones de la materia, que influyó en el origen de la ciencia química. Tuvo como fines principales la búsqueda de la piedra filosofal y de la panacea internacional.

[11] El Diccionario de la Real Academia Española ofrece varias definiciones de ésta, de las cuales la más completa se lee como el conjunto de doctrinas teosóficas basadas en la Sagrada Escritura, que a través de un método esotérico de interpretación y transmitidas por vías de iniciación, pretendía revelar a los iniciados doctrinas ocultas acerca de Dios y del mundo. Otra de las definiciones vertidas en el mismo diccionario se refiera a ella como un cálculo supersticioso para adivinar una cosa.

[12] Según Pastor y Sarasa, "El nacimiento del lenguaje escrito en sus formas más primitivas (ideográficas y jeroglíficas) puede considerarse a su vez como el origen de la criptografía, puesto que eran muy pocos los que podían leer o interpretar los símbolos utilizados." Pastor, J., Sarasa, M. Criptografía digital. Fundamentos y aplicaciones. Prensas Universitarias de Zaragoza. Zaragoza. 1998. pp. 567.

[13] David Kahn lo señala como el criptograma más antiguo del que se tiene conocimiento. Kahn, D. The Code Breakers, The Comprehensive History of Secret Communication from Ancient Times to the Internet. Scribner. New York. 1996. 2nd Edition. pp 71.

[14] Kahn, D. Op. Cit. Pp 75.
Stern J. Op. Cit. pp 22.

[15] Kahn, D. Op. Cit. pp. 74.

escitalo, el mensaje quedaba claro cuando se enrollaba la tira de papel alrededor de un rodillo de longitud y grosor adecuados.

En tiempos de Roma, se sabe que Julio César empleaba un código secreto que consistía en sustituir cada letra del mensaje por otra que en el alfabeto estuviese a tres posiciones de ella. Posteriormente, el escritor Aulio Gelio sugirió la idea de que en realidad Julio César recurría a procedimientos más complicados. El emperador Augusto, sobrino de Julio César, utilizó un sistema similar, pero Suetonio registra que simplemente sustituía una letra del alfabeto por la siguiente, salvo en el caso de la X (la última letra del alfabeto romano), a la que sustituía por AA.[16]

Uno de los más famosos manuscritos con contenido presuntamente mágico, el papiro *Leiden*, descubierto en Tebas y escrito en el siglo III en griego y más tarde en forma demótica[17], una versión ampliamente simplificada de jeroglíficos empleados para cifrar fragmentos cruciales de recetas y conjuros importantes. Por ejemplo, en una sección se explica cómo enfermar a un hombre con una enfermedad incurable de la piel, el papiro usaba signos secretos para cifrar las palabras "enfermedad de la piel" y nombres de variedades de lagartija: "Si desea enfermar a alguien con una enfermedad de la piel que no pueda ser curada, una lagartija *hantous* y otra lagartija *hafleele,* se cocinan con aceite, y se baña al hombre con la poción"[18].

Son de especial relevancia los desarrollos en materia de cifrado y criptoanálisis alcanzados por los árabes. Las revelaciones de Mahoma fueron registradas por varios escribas mientras él aun vivía, pero solo como fragmentos y correspondió a Abū Bakr, primer califa[19] del Islam, dar inicio a la compilación en un texto único. El trabajo fue continuado por Omar, el segundo califa, y su hija Hafsa, y fue eventualmente culminado por Uthmān, el tercer califa. Cada revelación se convirtió en uno de los 114 capítulos del Corán. El califa gobernante era responsable de llevar a cabo el trabajo del profeta, preservando y difundiendo su palabra. Durante el gobierno de los primeros cuatro califas, el Islam se extendió hacia la mitad del mundo conocido. En 750, después de un siglo de consolidación, el ascenso de la dinastía Abbasid trajo consigo la época dorada de la civilización islámica. Entre otras áreas del saber, son innumerables sus aportes a las matemáticas. Los califas de la dinastía Abbasid parecían menos interesados que sus predecesores en la expansión territorial y se concentraron en el establecimiento y organización de su sociedad. Menores impuestos impulsaron los negocios y posibilitaron un crecimiento comercial e industrial, además que una serie de estrictas leyes redujo la corrupción y llevó seguridad a la ciudadanía. Todo esto obtuvo soporte en comunicaciones seguras a partir de del cifrado de documentos, que era utilizada tanto para asuntos de gobierno como comerciales, a tal grado que está documentado el hecho de que funcionarios de gobierno protegían los registros de impuestos haciendo uso rutinario de procedimientos criptográficos. Llegaron a elaborarse manuales administrativos que daban instrucciones al respecto, tales como el *Adab al-Kuttāb* o "Manual de los secretarios" que data del siglo X, que incluye secciones completas dedicadas a la criptografía.

[16] Kahn, D. Op. Cit. pp. 84.
[17] El Diccionario de la Real Academia Española indica que se trata de un género de escritura cursiva empleado por los antiguos egipcios para diversos actos privados, así como también una variedad de la lengua griega moderna.
[18] Kahn, D. Op. Cit. pp. 91.
[19] Según el Diccionario de la Real Academia Española el término deriva del árabe *jalifa* que significa sucesor o lugarteniente. Era el título de los príncipes sarracenos que, como sucesores de Mahoma, ejercieron la suprema potestad religiosa y civil en Asia, África y España.

Los árabes llegaron a familiarizarse con formas de cifrado basadas en sustituciones monoalfabéticas, que en realidad no constituyen recursos extremadamente relevantes por su fortaleza en la historia de la criptografía. Sin embargo, además de estas técnicas, se dotaba a escolares de conocimientos y procedimientos que permitían romper ciertos esquemas de cifrado, debido a lo que puede considerárseles los creadores del criptoanálisis. Sobre esto hay que remarcar el hecho de que el criptoanálisis solo pudo ser inventado cuando una civilización alcanzó un nivel de escolaridad lo bastante sofisticado en disciplinas tan diversas como las matemáticas, la estadística y la lingüística, así como a la formación teológica.

Los árabes llegaron a dominar las bases del criptoanálisis basado en la observación de la frecuencia de letras. Aunque no se sabe exactamente quien fue el primero en emplearlo, las descripciones más antiguas de la técnica datan del siglo IX y se atribuyen al científico Abū Yūsūf Ya'qūb ibn Is-hāq ibn as-Sabbāh ibn 'omrān ibn Ismail al-Kindi. [20].

Como puede notarse, por cuestiones religiosas, los árabes avanzaron más en la creación y divulgación de las ciencias, la criptografía y el criptoanálisis no habría de ser la excepción. Mientras tanto, los reinos cristianos estaban inmersos en el apogeo del oscurantismo.

Durante la Edad Media e incluso durante el Renacimiento, la criptografía servía a menudo para propósitos mágicos. Un manuscrito compilado en Nápoles entre 1473 y 1490 por Arnaldus de Bruxella usa cinco líneas de cifrado para encubrir la parte más relevante del procedimiento para hacer la piedra filosofal. La asociación entre magia y criptografía se vio reforzada por varios aspectos. Símbolos misteriosos eran usados en campos tales como la astrología y la alquimia y también para fines criptográficos. Palabras cifradas y encantamientos tales como *abracadabra* parecían carecer de sentido, pero en realidad portaban mensajes presumiblemente importantes entre quienes los usaban. Por otra parte, otro factor importante que abonó a esta confusión fue la propagación de la curiosidad que suscitaba la cábala tanto entre judíos como cristianos[21].

Se sabe que San Bernardino evitaba la regularidad de los signos (con lo que el criptoanálisis por el método de las frecuencias no era efectivo) sustituyendo letras por varios signos distintos, así tenía un símbolo para cada consonante, usaba tres signos distintos para cada una de las vocales y utilizaba signos sin ningún valor.

Son particularmente interesantes los aportes de Roger Bacon (1210-1214), monje, filósofo, sabio, científico y teólogo inglés: es el único escritor de la Edad Media que no se limitó a usar la criptografía sino que llegó a describirla en su obra *Secret Works of Art and the Nulity of Magic*. Listó varios métodos de cifrado, entre ellos, algunos que usan sólo consonantes, expresiones figuradas, letras de alfabetos exóticos, caracteres inventados, taquigrafía, figuras mágicas y hechizos[22].

Geoffrey Chaucer, el poeta inglés, escribió el libro *The Equatorie of the Planetis*, el cual contiene varios pasajes cifrados hechos de letras, dígitos y símbolos, en los cuales, por ejemplo, la letra a es representada por un símbolo parecido a la letra V y b por uno parecido a la letra alfa.

[20] Singh, S. The code book. The Science of Secrecy from Ancient Egypt to Quantum Cryptography. Anchor books. New York. 1999. pp. 14-17.
[21] Kahn, D. Op. Cit. pp. 91-92.
[22] Kahn, D. Op. Cit. pp. 90.

En la Edad Media y el Renacimiento, los criptólogos eran a menudo matemáticos, los períodos siguientes los reemplazarían con frecuencia por militares, lo cual sin duda, no se debía más que a una cuestión de organización social.

Los duques de Sforza, gobernantes oligárquicos de Milán, también se sirvieron de la criptología. Uno de sus secretarios, Cicco Simonetta escribió uno de los primeros tratados dedicados enteramente al criptoanálisis: el *Liber Zifrorum*, en julio de 1474. En él establecía trece reglas para resolver cifrados de sustitución monoalfabética en los que se mantenían divisiones entre palabras. El manuscrito comenzaba diciendo: "el primer requisito es ver si el documento se encuentra en latín o en lengua vernácula, este puede determinarse de la siguiente manera: ver si las palabras del documento en cuestión tienen más o menos de cinco diferentes terminaciones; si hay cinco o menos puede concluirse que esta en lengua vernácula...".

Otras cortes europeas también tenían criptoanalístas a su servicio. En Francia, Philibert Babou, señor de Bourdaisière, que ocupaba el cargo de primer secretario de estado resolvía despachos interceptados para Francisco I. Babou fue capaz de resolver criptogramas de lenguas que no sabía hablar, o que entendía muy poco, como el español, italiano o alemán[23].

El conocimiento de la criptología llegó hasta Iberia en la época en que los Reyes Católicos expulsaron a los moros y unificaron el país para comenzar a erigirlo como potencia mundial. Los primeros sistemas, introducidos en 1480 por Miguel Pérez Alzamán, transformaban el texto claro en numerales romanos, el resultado era tan difícil de decodificar que algunos descifradores marginaron las notas con frases como: "sin sentido", "imposible", "no puede ser entendido", "ordene al embajador enviar otro despacho".

Se considera a Johannes Trithemius como el padre de la criptografía moderna. Nacido en Trittenheim, una pequeña ciudad alemana de la que le viene el nombre, era un abad benedictino que se interesó por las ciencias naturales y se ganó fama de mago lo que hizo que el mismo terminara siendo declarado hereje por la Inquisición. Este religioso escribió en 1530 *Poligrafía*, uno de los libros más importantes que abordan el tema. Trithemius introdujo el concepto de tabla ajustada, en el cual el alfabeto normal es permutado para codificar los mensajes.

El noble de Brescia, Giovan Battista Belaso, publicó en 1553 El auténtico modo para escribir en cifra. También había descrito los cifrarios polialfabéticos. Belaso y Blaise de Vigenère ofrecieron un sistema de autoclave para incrementar la seguridad de los criptogramas.

Felipe II ascendió al trono de España en 1556 a los 29 años de edad, y notificó a su tío Fernando I de Hungría, que había decidido cambiar el cifrado usado durante el reinado de su padre Carlos V porque éste había caído en desuso, y podía haberse comprometido. Así, Felipe II entregó a su emisario Juan de Moreo, un sistema que contenía alrededor de 400 grupos de código. François Viète, al servicio de Enrique IV de Francia, trabajó entre 1589 y 1590 para romper el acertijo, luego brindó una solución completa a su rey. Luego Felipe II descubrió que un mensaje que él consideraba irrompible estaba comprometido y se quejó ante el Vaticano argumentando que Enrique IV había recurrido a la magia negra para romper su código. El Papa estaba mejor

[23] Es posible resolver un criptograma en un lenguaje que no se conoce, partiendo de que ese desconocimiento significa sólo que no se comprende el significado de las palabras. Kahn, D. Op. Cit. pp 110-111.

informado, pues su propio criptólogo Giovanni Battista Argenti también había roto el código, con lo que el Felipe II se convirtió en el hazmerreír de Europa[24].

En Francia, el cardenal Richelieu (1585–1642), quien fuera Primer Ministro de Luis XIII, tuvo a su servicio al primer criptólogo de tiempo completo Antoine Rossignol, quien se convirtió en un importante bastión en la lucha contra los hugonotes y las potencias europeas de la época.

La descendencia de Antoine Rossignol también habría de consagrarse en trabajos criptográficos al servicio de la Corona francesa. Por ejemplo su hijo Bonaventure y su nieto Antoine-Bonaventure también hacía criptoanálisis para corona en tiempos de Luis XIV y Luis XV.

En el siglo XVI, Girolamo Cardano, físico y matemático italiano, utilizó el método de la tarjeta con agujeros perforados, que se debía colocar sobre un texto para poder leer el mensaje cifrado, además fue el creador del primer texto que versaba sobre la teoría de la probabilidad. Inventor del primer e imperfecto sistema de autoclave, en el cual cada mensaje de texto claro tiene la clave para cifrarse a sí mismo.

Cardano publicó 131 libros y 111 manuscritos, en los cuales discutía sobre matemáticas, astronomía, astrología, física, ajedrez, apuestas, la inmortalidad del alma, curas milagrosas, dialectos, la muerte, gemas y colores, venenos, aire, agua, sueños, orina, dientes, música, moral y sabiduría. Nunca publicó un libro sobre criptología, pero sí abordaba el tema en sus dos libros más difundidos: *De Subtilitate* y *De Returm Varietate*, en los cuales describía los métodos clásicos de cifrado de la antigüedad.

Blaise de Vigenère publicó en 1586 su *Traicté des Chiffres* donde recoge los distintos métodos utilizados en su época, el método Vigenère es un método clásico de cifrado por sustitución que utiliza una clave, del cual se dan más detalles en apartados posteriores. Fue también en 1586 que Thomas Phelippes rompió el cifrado de la Reina Maria de Escocia.

Sir Francis Bacon (1561-1626) inventó el uso de dos tipos de letra para transportar un mensaje secreto. Él describió su método en la versión latina *De dignitate et augmentis scientiarum* (1623) de un libro titulado *Proficience and Advancement* que había publicado en 1605.Su método nunca logró gran importancia práctica[25].

Carlos I de Inglaterra usó en el siglo XVII códigos de sustitución silábica. Napoleón, en sus campañas militares y en los escritos diplomáticos, usó los llamados métodos Richelieu y Rossignol y para evitar la regularidad de los símbolos asignaba números a grupos de una o más letras.

Las continuas guerras e innumerables confabulaciones y conspiraciones en Europa dieron lugar a que las cortes de las grandes potencias europeas organizaran sus correspondientes y departamentos expertos de criptoanálisis: son famosos el siniestro *Cabinet Noir*, de París, y la *Geheime Kabinets - Kanzlei*, de Viena.

El siglo XVIII no fue una época de grandes avances criptográficos. El telégrafo, inventado por Samuel Morse a principios del siglo XIX, y la aparición de la radio revolucionaron las

[24] Bauer, F. Decrypted Secrets. Methods and Maxims of Cryptology. Springer Verlag. New York. 2002. pp. 68.
Singh, S. Op. Cit. pp. 28-29.
[25] Bauer, F. Op. Cit. pp. 9.

comunicaciones y obligaron a la criptografía a desarrollarse como ciencia. Ambos medios eran fáciles de interceptar lo que dio lugar a que en los ámbitos militares y diplomáticos se buscaran nuevas maneras de mantener en secreto importantes mensajes.

En el año de 1857, en Inglaterra, se presentó un nuevo sistema de escritura secreta "adaptado para telegramas y postales", consistía en una tarjeta de 4x5 pulgadas con el alfabeto impreso en color negro y rojo. Su creador fue el almirante Sir Francis Beaufort, quien dio origen a su cifrado utilizando la escala mediante la cual los meteorólogos indicaban la velocidad del viento, con números desde 0 (calmado) hasta 12 (huracán). Después de la muerte del almirante, fue su hermano quien hizo público el criptosistema.

Usa un cuadro alfabético esencialmente similar al de Vigenère, pero se diferencia de él al repetir el alfabeto. El sistema había sido originalmente propuesto por Giovanni Sestri casi 150 años que Beaufort, en un libro publicado en Roma en 1710, que fue ampliamente ignorado. Si embargo, bajo el nombre de Beaufort, este cifrado se convirtió en un estándar del repertorio de la criptografía, pese a que su importancia teórica no es significativa.

En el siglo XIX se utiliza ampliamente el método de transposición, consistente en la reordenación según distintos criterios de los símbolos del mensaje. Auguste Kerckhoffs, lingüista holandés naturalizado francés, indica las reglas que a su juicio debía cumplir un buen sistema criptográfico:

- El sistema debe ser materialmente, y matemáticamente, indescifrable;
- El hecho de que caiga en manos del enemigo no debería posibilitar el descubrimiento del secreto;
- La clave debe poder ser comunicada y retenida sin el envío de notas escritas, así como cambiada y modificada según el criterio de las partes;
- Debe ser aplicable a la correspondencia telegráfica;
- Debe ser portátil, y su manipulación o funcionamiento no debe exigir el concurso de muchas personas;
- Finalmente, es necesario, vistas las circunstancias requeridas para su aplicación, que el sistema sea de fácil uso, que no demande ni tensión de ánimo ni el conocimiento de una larga serie de reglas a observar.[26]

De lo anterior puede considerarse que la seguridad, la simplicidad y la rapidez son los conceptos clave de un sistema criptográfico. Sin embargo, la idea más relevante de las expresadas por Kerckhoffs es la que enuncia que mientras la clave permanezca secreta, no importa tanto si el algoritmo es público, pues es una idea que aun conserva validez. Otro teórico del siglo XIX llamado Wilhelm Kasiski, oficial prusiano, revolucionó la criptología casi sin darse cuenta y abrió las puertas a la criptología moderna en una obra que publicó en 1863 titulada Les chiffres et l'art du décryptement, que expone por primera vez métodos estructurados de criptoanálisis, en particular su método de decriptamiento de sustituciones polialfabéticas.[27]

Charles Babbage diseñó en 1823 la Difference Engine No. 1, una calculadora mecánica que constaba de alrededor de 25,000 piezas de precisión, además de dedicarse al diseño de máquinas de este tipo, Babbage tenía sólidos conocimientos de criptoanálisis, a tal grado que fue capaz de romper el cifrado de Vigenère, para lo cual Babbage se valió del análisis de frecuencia. Kasiski también habría de romper dicho esquema de cifrado basándose en el método de incidencia de las coincidencias en que publicó en la obra de 1863 a la que se hizo referencia antes. La

[26] Kerckhoffs, A. *La cryptographie militaire.* Journal des Sciences Militaires. Janvier 1883.
[27] Stern, J. La science du secret. Éditions Odile Jacob. Paris. 1998. Pp. 28-29.

repetición de un determinado grupo de letras en el criptograma, proveniente de un mismo grupo de letras en el texto claro, tiene lugar a una distancia múltiplo de la longitud de la palabra clave. Estudiando estas repeticiones puede determinarse la longitud de la palabra clave, conocida dicha palabra, el criptograma se descompone en criptogramas sencillos correspondientes a cifrados de César.

Durante la Primera Guerra Mundial, el trasatlántico Lusitania fue hundido por los alemanes en 1915, pereciendo más de 1,200 personas. Este hecho polarizó la opinión pública norteamericana apoyando la entrada de los Estados Unidos en la guerra. El presidente T. Woodrow Wilson, quien había fomentado activamente la no beligerancia de su país, se oponía. Sin embargo, el 17 de enero de 1917 los servicios secretos ingleses interceptaron un telegrama cifrado del ministro de Asuntos Exteriores de Alemania, Arthur Zimmermann, dirigido al conde Heinrich A. von Bernstorff, embajador alemán en Washington. El mensaje se había cifrado por medio del código 0075, un diccionario de lista doble de 10,000 palabras o frases que los ingleses ya habían conseguido descifrar parcialmente. En él se hacía referencia a otro telegrama que debía enviarse a México. Los ingleses consiguieron interceptar este segundo telegrama y observaron que se había cifrado con otro código, el 13040, que constaba de 25,000 palabras, esto representó un grave error criptográfico ya que permitió a los ingleses descifrar el mensaje. El resultado fue sorprendente: si México declaraba la guerra a los Estados Unidos para recuperar sus antiguos territorios de Nuevo México, Arizona y Texas, Alemania le apoyaría. Wilson se vio forzado a entrar en guerra contra Alemania.

2.1.1 Métodos artesanales

2.1.1.1 Atbash

Método hebreo de alrededor de los años 600–500 A.C., que consistía en tomar cada letra, calcular el número de lugares que lo separan de la primera letra del alfabeto y reemplazarla con una letra que se encuentra en la misma distancia del final del mismo.

En alfabeto español equivale a reemplazar:

- La letra "a", al principio alfabeto, por la letra "z";
- La letra "b" se cambia por la letra "y".

El nombre Atbash proviene de las dos primeras y las dos ultimas letras del alfabeto hebreo, combinadas de la siguiente forma: *aleph, taw, beth, shin*. Un ejemplo de Atbash aparece en Jeremías 25:26 y 51:41, donde el nombre “Babel” es reemplazado por la palabra “Sheshach”, la primera letra de Babel *beth,* segunda letra del alfabeto hebreo, y es reemplazada por *shin,* la penúltima; en hebreo la segunda letra de Babel es también *beth,* y nuevamente es reemplazada por *shin;* la última letra de Babel es lamed, duodécima letra del alfabeto hebreo y es reemplazada por *kaph,* que es la que se encuentra a doce posiciones de la última letra del alfabeto[28].

[28] Kahn, D. Op. Cit. pp 76-80.
Singh, S. Op. Cit. pp 26.

aleph	beth	gimel	dalet	he	waw	zayin	heth	teth	yod	Kaph
א	ב	ג	ד	ה	ו	ז	ח	ט	י	כ
ל	מ	נ	ס	ץ	ם	צ	ק	ר	ש	ת
lamed	mem	Nun	samekh	ayin	pe	sadhe	qoph	resh	shin	Taw

Tabla 2.1: Alfabeto hebreo

2.1.1.2 Método de Escítalo

En la Grecia clásica, los *éforos* (gobernantes) espartanos transmitían sus instrucciones a sus *estrategas* (generales) utilizando un bastón, el escitalo, siglo IV A.C. El historiador griego Plutarco describe la el escitalo o *scitala espartana* como una vara de la que se preparaban dos o más bastones idénticos. Las órdenes se escribían en una tira de pergamino o papiro enrollada a lo largo del bastón. Desenrollada, solamente contenía una sucesión de letras inconexas que se enviaba al destinatario, para poder leer el mensaje éste debía tener en su poder una copia del bastón. Al colocar de nuevo la cinta en el bastón aparecía el mensaje[29].

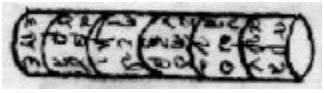


Figura 2.2: Escitalo

2.1.1.3 Método de César

El algoritmo de César, llamado así porque es el que empleaba Julio César para enviar mensajes secretos, es uno de los algoritmos criptográficos más simples. Consiste en sumar 3 números de orden de cada letra. De esta forma a A le corresponde D, a Be la E, y así sucesivamente.
Para descifrar basta con restar 3 al número de orden de las letras del criptograma[30].

2.1.1.4 El Kama Sutra

El libro erótico más famoso de Vãtsyāyana, el Kama Sutra, lista la escritura secreta como una de las 64 artes o yogas que las mujeres deben saber y practicar, es la número 45 de una lista que inicia con la música vocal, pasando por prestidigitación, solución de rompecabezas verbales, y ejercicios de poesía enigmática. Este arte es llamado *mlecchita-vikalpa*, una de sus ramas llamada *kautiliyam* en la cual la sustitución de letras estaba basada en relaciones fonéticas, por ejemplo, las vocales se convertían en consonantes.

El texto claro en la mayor parte de las secciones cifradas estaba escrito en griego, y el alfabeto cifrado consistía básicamente en símbolos de letras griegas.

[29] Kahn, D. Op. Cit. pp 82.
Singh, S. Op. Cit pp 8-9.
[30] Brisson, R., Théberge, F. *Un aperçu de l'histoire de la cryptologie*. Descargado de http://collection.nlc-bnc.ca/100/200/301/cse-cst/overview-f/musee.pdf el 7 de diciembre de 2003.

2.1.1.5 Nomenclator

A petición de Clemente VII, Gabrieli di Lavinde que era uno de sus secretarios, compila un conjunto de llaves individuales para un alfabeto de sustitución en un código pequeño. Su colección de llaves (la más antigua existente en la civilización occidental moderna) incluía varias que combinaban elementos para código y cifrado. Cada clave correspondía a un pequeño repertorio de una docena o más palabras comunes o nombres equivalentes de dos letras. El *Nomenclator* unía alfabetos de sustitución y listas de códigos de palabras, sílabas y nombres equivalentes.

Esta clase de sistema criptográfico permaneció en uso entre diplomáticos y algunos civiles durante los siguientes 450 años[31].

2.1.1.6 Método de Alberti

Leon Battista Alberti (1402-1472), uno de los más famosos exponentes del Renacimiento italiano, escribió un tratado titulado *Modus scribendi in ziferas*, donde describe, entre otras cosas, unos discos con los que se podían cifrar mensajes. Este hombre fue secretario de claves de la Curia Vaticana, otra potencia de la época, su contribución fue tan importante que ha merecido el título de "padre de la criptografía occidental".

2.1.1.7 Método de Trithemius

En 1508 inició la publicación de su *Polygraphiæ*, obra en seis volúmenes que presentaba una colección de palabras en latín que codificaba a las letras del alfabeto, lo que se conoció posteriormente como el código *Ave Maria*.

A	Deus
B	Creator
C	Conditur
D	Opisex
E	Dominus
F	Dominator
G	Consolator
H	Arbiter

A	clemens
B	clementissimus
C	pius
D	pijssimus
E	magnus
F	excelsus
G	maximus
H	optimus

Tabla 2.2 Código Ave Maria

Así, por ejemplo la palabra ACECHA podría cifrarse como: DEUS PIUS DOMINUS MAGNUS CONDITOR OPTIMUS CLEMENS

2.1.2 Sistema de autoclave

Girolamo Cardano fue el creador de éste sistema, en el cual se utiliza el mensaje claro para generar su propia clave. La principal observación es que la clave es función del texto claro y que además la clave puede cambiar con cada mensaje.

[31] Bauer, F. Op. Cit. pp. 68-70.
Kahn, D. Op. Cit. pp. 107.

2.1.2.1 Método de Porta

Este sistema fue puesto en marcha en 1563 por Giovanni Battista da Porta. Su método es descrito por medio de la Tabla 2.3 Requiere una palabra clave cuyas letras formen letras clave. La primera columna, que contiene pares de letras secuencialmente ordenadas del alfabeto, contiene la componente de la palabra clave.

La fila de arriba, contiene la componente primaria del texto claro. Su asociación permite una sustitución recíproca para una letra clave en particular. Supongamos una letra clave, si la letra del texto claro figura en la fila de arriba, se le sustituye por la letra que aparece en la intersección con la columna donde se encuentra la letra en claro y la fila donde se encuentra la letra clave. Si la letra del texto claro no forma parte de la fila de arriba, se le busca en la fila donde se encuentra la letra clave y se le sustituye por la letra correspondiente.

	a	**b**	**c**	**d**	**e**	**F**	**g**	**h**	**i**	**j**	**k**	**l**	**M**
AB	n	o	p	q	r	S	t	u	v	w	x	y	Z
CD	z	n	o	p	q	R	s	t	u	v	w	x	Y
EF	y	z	n	o	p	Q	r	s	t	u	v	w	X
GH	x	y	z	n	o	P	q	r	s	t	u	v	w
IJ	w	x	y	z	n	O	p	q	r	s	t	u	V
KL	v	w	x	y	z	N	o	p	q	r	s	t	U
MN	u	v	w	x	y	Z	n	o	p	q	r	s	T
OP	t	u	v	w	x	Y	z	n	o	p	q	r	S
QR	s	t	u	v	w	X	y	z	n	o	p	q	R
ST	r	s	t	u	v	W	x	y	z	n	o	p	Q
UV	q	r	s	t	u	V	w	x	y	z	n	o	P
WX	p	q	r	s	t	U	v	w	x	y	z	n	O
YZ	o	p	q	r	s	T	u	v	w	x	y	z	N

Tabla 2.3. Tabla de Porta

Cifremos ENVIAR MEDICINAS utilizando como palabra clave la palabra GUAZAPA.

Texto claro: ENVIAR MEDICINAS
Palabra clave: GUAZAPA

	A	**b**	**c**	**d**	**e**	**F**	**g**	**h**	**i**	**j**	**k**	**l**	**m**
GH	X	y	z	n	o	P	q	r	s	t	u	v	w
UV	Q	r	s	t	u	V	w	x	y	z	n	o	P
AB	N	o	p	q	r	S	t	u	v	w	x	y	Z
YZ	O	p	q	r	s	T	u	v	w	x	y	z	N
AB	N	o	p	q	r	S	t	u	v	w	x	y	Z
OP	T	u	v	w	x	Y	z	n	o	p	q	r	S
AB	N	o	p	q	r	S	t	u	v	w	x	y	Z

```
Palabra clave:          GUAZAPAGUAZAPAG
Texto claro:            ENVIARMEDICINAS
Texto cifrado:          OKIWNLZOTVQVTNI
```

2.1.2.2 Método de Vigenère

Este método de cifrado es el fruto del trabajo de Blaise de Vigenère, un francés que vivió entre 1523 y 1596. Al parecer que el método fue puesto a punto por Vignère durante sus visitas al Vaticano.

El principio de este método consiste en utilizar una sustitución alfabética diferente para cada posición, lo cual torna el análisis de frecuencias un poco menos atractivo. Una palabra clave es utilizada y escrita en muchas ocasiones sobre el texto claro tal como en el método de Porta. En el ejemplo que sigue, la palabra clave es VOLCAN. Para cifrar, se elige la fila de la Tabla 3 que corresponde a la letra apropiada de la palabra clave y se opera una sustitución alfabética con la letra situada en la intersección de la columna correspondiente a ésta y de la fila correspondiente a la letra del texto claro. El cifrado del texto claro se efectúa entonces para tantas sustituciones diferentes como letras hay en la palabra clave.

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

Tabla 2.4 Tabla de Vigenère

```
Palabra clave:          VOLCANVOLCANVOLC
Texto claro:            SALIRDESANMIGUEL
Texto cifrado:          NOWKRQZGLPMVBIPN
```

Para este método, el destinatario debe conocer la palabra clave y la tabla de cifrado. Esta tabla puede ser tan simple como la antes presentada. El proceso de descifrado se completa simplemente procediendo a la inversa.

2.1.2.3 Método de Playfair

El cifrado de Playfair en realidad fue inventado por Charles Wheatstone, para comunicaciones telegráficas secretas en 1854, no obstante se le atribuye a su amigo el científico Lord Playfair.

Utilizado por el Reino Unido en la Primera Guerra Mundial, este sistema consiste en separar el texto en claro en diagramas y proceder a su cifrado de acuerdo a una matriz alfabética de dimensiones 5 X 5 en la cual se encuentran representadas las 26 letras del alfabeto ingles, aunque para una mayor seguridad se puede agregar una palabra clave. La clave se coloca al comienzo de la matriz quitando las repeticiones y a continuación el resto de las letras del alfabeto.

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

Tabla 2.5 Matriz alfabética

A continuación se presenta una versión simplificada de este método:

- Llenar una tabla de dimensiones 5x5 con letras del alfabeto en pares. Esta puede ser complementada con una palabra clave inscribiendo consecutivamente en la tabla la primera aparición de cada letra de la palabra clave seguida de las otras letras del alfabeto. La Tabla 3 se ha construido a partir de la palabra clave PERRO agrupando I y J.

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

Tabla 2.6 Matriz alfabética de tipo Playfair

- Escribir el texto claro por grupos de dos letras. Si un par contiene la misma letra dos veces, se agrega una *letra complementaria* (como la X) entre ellas. Por ejemplo, CARRETERA BLOQUEADA se escribe CA RX RE TE RA BL OQ UE AD AX.

- La letra complementaria es X y es utilizada en dos ocasiones, la segunda de las cuales fue para completar el último par del mensaje.

- Para cada par de letras, la Tabla 2.6 es utilizada como sigue:

    - Si las letras están en la misma columna, cada una de ellas es reemplazada por la letra situada inmediatamente abajo; si una letra está debajo de la columna, es reemplazada por la primera de la columna;

- Si las letras están en la misma fila, cada una es reemplazada por la letra situada inmediatamente a su derecha; si una de las letras es la última de la fila, es reemplazada por la primera de esa misma fila.

- Si las dos letras no se encuentran ni en la misma fila ni columna, se les reemplaza de la manera siguiente: la primera letra cifrada es obtenida tomando la letra situada en la intersección de la fila que contiene la primera letra del texto claro; la segunda letra cifrada se obtiene tomando la intersección de la columna que contiene la primera letra del texto claro y de la fila que contiene la segunda letra del texto claro.

Texto claro	Caso	Texto cifrado
CA	Diferente fila y columna	GE
RX	Diferente fila misma columna	DR
RE	Misma fila diferente columna	OR
TE	Diferente fila y columna	QO
RA	Misma fila diferente columna	OP
BL	Diferente fila y columna	FH
OQ	Diferente fila y columna	ET
UE	Diferente fila y columna	QA
AD	Diferente fila y columna	RG
AX	Diferente fila y columna	RZ

Tabla 2.7 Proceso de cifrado de tipo Playfair

Las etapas de cifrado del texto claro del ejemplo anterior son presentadas en el cuadro siguiente. El texto cifrado es en seguida escrito como una secuencia continua de letras: GEDRORQOOPFHETQARGRZ.

2.1.2.4 Caracterización de la era artesanal

Entre los orígenes de la criptografía y la primera guerra mundial, los métodos artesanales fueron mejorando lentamente, permaneciendo bastante vulnerables, debido a dos factores, el primero, la pobreza del entorno tecnológico, y el empirismo reinante.

Son notorios los límites de las defensas que los criptógrafos oponían a los criptoanalistas de la era artesanal. Dichos límites estaban ligados a la restringida capacidad física de cifrar y descifrar sin el auxilio de documentos específicos, cuya seguridad seguía siendo problemática. Estos inconvenientes no carecen de consecuencias sobre el desarrollo de los principios mismos de la criptografía; es así que los métodos polialfabéticos, en los que la clave evoluciona durante el proceso de cifrado, no pudieron imponerse en ausencia de máquinas (pese a haber sido ya vislumbrados), pues el menor error se podía propagar hasta el final del texto volviéndolo ininteligible también al destinatario[32].

2.1.2.5 Criptosistema de Vernam

El método de Vigenère fue llevado a su máxima extensión lógica en 1935 por el criptógrafo estadounidense Gilbert S. Vernam. Este investigador demostró que para que el cifrado de Vigenère fuera seguro no solamente era necesario que la clave de cifrado fuese más larga que el

[32] Stein, J. Op. Cit. pp. 38-39.

mensaje, sino que además debería ser utilizado una sola vez. Bajo estas condiciones el cifrado de Vernam (también denominado *one time pad*) es un cifrado perfecto, tal como demostró Shannon posteriormente. Es conveniente hacer notar en este punto que el cifrado perfecto de Vernam es de difícil e incluso imposible implementación práctica con la tecnología actual. A pesar de ello, es el que sirve como modelo de referencia para muchos otros procedimientos de cifrado desarrollados en la criptografía moderna, en particular para los de flujo[33].

2.1.2.6 ADFGVX

Este cifrado, inventado por Fritz Nebel (1891-1967), se instaló en el frente occidental alemán que estaba bajo el mando del General Erich Ludendorff, para la transmisión inalámbrica. Este método fue de los primeros en unir la transposición y la sustitución dos procesos que producen, en la terminología actual, difusión y confusión[34].

El cifrado inicia al dibujar una cuadrícula de 6x6 las filas y las columnas se encabezan con las letras ADFGVX, se llenan los 36 cuadros con las 26 letras del alfabeto anglosajón y 10 dígitos, el modo de ordenar letras y números en la cuadrícula forma parte de la clave y necesita ser comunicada al receptor del mensaje.

	A	**D**	**F**	**G**	**V**	**X**
A	0	q	9	z	7	c
D	m	u	1	h	f	2
F	4	8	w	n	r	g
G	l	6	v	t	p	a
V	y	3	d	5	e	k
X	j	s	i	o	b	x

Tabla 2.8 Matriz ADFGVX

Por ejemplo:

La primera fase para realizar el cifrado consiste en tomar cada letra del mensaje claro y sustituirla por las letras correspondientes a su fila y columna.

Por ejemplo el número 5 sería sustituido por las letras VG y la j por el par de letras XA.

El mensaje claro: ENVIEN MUNICIONES

E	N	V	I	E	N	M	U	N	I	C	I	O	N	E	S
VV	FG	GF	XF	VV	FG	DA	DD	FG	XF	AX	XF	XG	FG	VV	XD

Hasta aquí solo se ha realizado la parte de sustitución.

[33] Pastor, J., Sarasa, M. Op Cit pp. 575.
[34] Bauer, F. Op. Cit. pp. 51 y 159.
Singh, S. Op. Cit. pp. 103-104.

La segunda fase es en la cual se utiliza la trasposición que depende de una palabra clave. Si la clave es WHISKY. Las letras de la clave se escriben en la cabecera de una cuadrícula. El texto que hemos cifrado antes se escribe por filas en dicha cuadrícula:

W	**H**	**I**	**S**	**K**	**Y**
V	V	F	G	G	F
X	F	V	V	F	G
D	A	D	D	F	G
X	F	A	X	X	F
X	G	F	G	V	V
X	D	0	0	0	0

Tabla 2.9 Matriz ADFGVX con palabra clave

Se añaden caracteres de relleno (00) para que el cuadro quede completo. Ahora las columnas de la cuadrícula se cambian de posición de modo que las letras de la clave queden en orden alfabético:

H	**I**	**K**	**S**	**W**	**Y**
V	F	G	G	V	F
F	V	F	V	X	G
A	D	F	D	D	G
F	A	X	X	X	F
G	F	V	G	X	V
D	A	A	A	X	A

Tabla 2.10 Matriz ADFGVX con palabra clave ordenada alfabéticamente

El resultado de leer la tabla 2.10, es el texto cifrado:

VFAFGDFVDAFAGFFXVAGVDXGAVXDXXXFGGFVA

El texto cifrado se transmitía usando clave Morse, ya que el resultado sólo consta de 6 letras.

La razón por la cual se eligieron las letras ADFGVX en lugar de ABCDEF, es porque las primeras son muy diferentes una de la otra al ser traducidas a la clave Morse, de esta forma, se minimizaba el riesgo de confusión durante la transmisión.

A	D	F	G	V	X
.-	-..	..-.	--.	...-	-..-

Tabla 2.11 Letras del alfabeto y clave Morse

2.1.3 Otros métodos

Los métodos de cifrado descritos hasta el momento son tales que el texto claro es combinado con una clave secreta, según un algoritmo específico para producir el texto cifrado. Existen, no obstante, varias otras formas de transmitir secretamente un mensaje: a decir verdad, estas no

forman necesariamente parte del dominio de la criptología. Por ejemplo, se puede utilizar tinta invisible sobre ciertos tipos de papel. El papel puede ser calentado o tratado con productos químicos a fin de exponer el mensaje secreto. Otro método es la miniaturización de la representación física de la información: pensemos en micropuntos o microfichas, por ejemplo. El cifrado no es, entonces, la única manera segura de transmitir un mensaje secreto, pero es práctica y fácil de utilizar, lo que explica su popularidad.

2.2 Era Técnica

Los sistemas manuales son a menudo lentos y trabajosos para el usuario, al estar basados en el empleo de lápiz y papel. Además, no permiten el uso de algoritmos complicados. Por ello métodos de cifrado más rigurosos y complejos que utilizan aparatos mecánicos fueron desarrollados. La sección que sigue ofrece una breve aproximación a los más célebres de estos aparatos.

2.2.1 Modelos de máquinas de cifrado[35]

2.2.1.1 Aparato de discos codificadores de Alberti

Este aparato se desarrolló en tiempos de la era artesanal, evidentemente no se le producía en serie, pero se le incluye en esta parte por ser arquetipo de muchos de los sistemas criptográficos mecánicos que surgirían durante la era técnica.

El primer aparato de discos codificadores fue inventado por León Battista Alberti en el siglo XV. Estaba formado por dos discos concéntricos de cuero, uno de estos discos era grande y fijo y el otro más pequeño y móvil. Estos discos estaban divididos en 24 partes radiales iguales. El disco exterior contenía las letras del texto claro en el orden siguiente:

{ A, B, C, D, E, F, G, I, L, M, N, O, P, Q, R, S, T, V, X, Z, 1, 2, 3, 4 }

Es decir, un número suficiente de letras del alfabeto para formar la mayoría de palabras latinas. El disco interior contenía la siguiente permutación del alfabeto latino:

{ m, r, d, l, g, a, z, e, n, b, o, s, f, c, h, t, y, q, i, x, k, v, p, et }

Este mecanismo, bastante simple en sí mismo, ilustra el ingenio de Alberti, quien combina por primera vez una sustitución polialfabética y el uso de un código.

Otro ejemplo clásico de aparato de disco codificador es el provisto por la *"Confederate Cipher Disk"*, fabricado en latón. Este aparato contiene también los discos exterior e interior, portando ambos las letras en orden alfabético. La primera idea era utilizar este mecanismo para el método de Vigenère.

2.2.1.2 El cilindro de Jefferson

El presidente estadounidense Thomas Jefferson (1743-1826) inventó un dispositivo de cifrado, aunque el primero en fabricarla en serie fue Étienne Bazeries en 1891.

[35] Brisson, R., Théberge, F. Op. Cit. pp. 10-16.
Las imágenes de las máquinas que se presentan en este apartado han sido tomadas del sitio web http://webhome.idirect.com/~jproc/crypto/menu.html

Los sistemas de cifrado anteriores a la II Guerra Mundial son considerados como clásicos, tienen en común que pueden ser empleados usando simplemente lápiz y papel, y que pueden ser criptoanalizados casi de la misma forma.

Este invento consiste en una serie de discos que giran alrededor del mismo eje con letras impresas del alfabeto, colocadas en diverso orden. El emisor va moviendo los discos hasta conseguir poner en línea las letras adecuadas a su mensaje. Entonces lo codifica transmitiendo las letras que hay en cualquier otra línea. El receptor, descifra el mensaje, tomando su propia rueda y pone las letras del código en orden y sólo tendrá que buscar la línea de letras con el mensaje recibido.

En la siguiente figura vemos el resultado de cifrar las palabras "secretword" es decir "mvdtswxhxr".

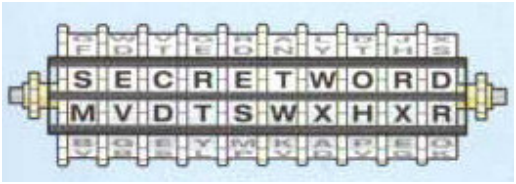


Figura 2.3 El cilindro de Jefferson

Los criptosistemas clásicos en la actualidad han perdido su eficacia, debido a que son fácilmente criptoanalizables empleando cualquier computadora, pero que fueron empleados con éxito hasta principios del siglo XX.

La criptografía clásica era una ciencia o más bien un arte secreto y casi exclusivo de los ámbitos oficiales de los ejércitos y cuerpos diplomáticos. Su utilización en otros ámbitos era también secreta, puesto que se practicaba en el comercio de alto nivel normalmente asociado con las clases gobernantes. Así como en el mundo de la magia y de la alquimia, con el objeto de transmitir conocimientos que permitiesen ejercer poder sobre los no iniciados.

2.2.1.3 El cilindro de ruedas codificadas M-94/CSP-488

Figura 2.4 Cilindro de ruedas codificadoras M-94

En 1922, el ejército de los Estados Unidos hizo fabricar el M-94, un aparato cilíndrico que contenía 25 anillos de un diámetro aproximado de 4 cm en aluminio sobre un pivote de alrededor de 10.5 cm de longitud. Este mecanismo permaneció en servicio hasta el principio de la Segunda

Guerra Mundial. También fue utilizado por la guardia costera y la FCC36 de los Estados Unidos. La marina estadounidense poseía una versión parecida llamada CSP-488.

2.2.1.4 La máquina ENIGMA[37]

En el año de 1923, un ingeniero alemán llamado Arthur Scherbius patento una de las mejores máquinas de cifrado, específicamente diseñada para facilitar las comunicaciones seguras. Se trataba de un instrumento de apariencia simple, parecido a una máquina de escribir. Quién deseara codificar un mensaje sólo tenía que teclearlo y las letras correspondientes al mensaje cifrado se irían iluminando en un panel. El destinatario copiaba dichas letras en su propia máquina y el mensaje original aparecía de nuevo.

El gobierno alemán adquirió todos los derechos sobre la máquina y la adaptó a sus necesidades, luego, ésta se convirtió en un estándar para los militares, agentes y policías secretos. Además de la máquina alemana ENIGMA, existieron otros dispositivos criptográficos basados en rotores, por ejemplo, SIGABA la máquina empleada por el ejército norteamericano, y las máquinas japonesas PURPLE y RED.

Figura 2.5 La máquina ENIGMA y sus rotores

[36] FCC: Federal Communications Comisión. Ente federal de los Estados Unidos encargado de las regulaciones a las telecomunicaciones.
[37] Sale, Tony (Translator). *The Bletchley Park translated Enigma Instruction Manual*. Traducido en 2001 del documento *Der Schlüssel M Verfahren M Allgemein* del Comando Supremo de la Marina Alemana, elaborado en 1940. Descargado de http://www.codesandciphers.org.uk/documents/officer/officer1.pdf el 8 de diciembre de 2003.

2.2.1.5 Las máquinas de Hebern

En las dos décadas anteriores a la Segunda Guerra Mundial, Edward H. Hebern (1869-1952) fue el primer inventor americano en hacer una contribución significante al desarrollo de máquinas de cifrado. Hebern inventó, en 1924, máquinas hechas de bronce de 3 y 5 ruedas codificadoras como prototipos para la marina estadounidense. Estas máquinas empleaban ruedas codificadoras cuyo tendido de cables interno podía ser fácilmente cambiado, además, estas ruedas codificadoras podían funcionar en el sentido de rotación convencional o en el contrario. Contrariamente, estas máquinas poseían infinidad de debilidades desde el punto de vista del criptoanálisis.

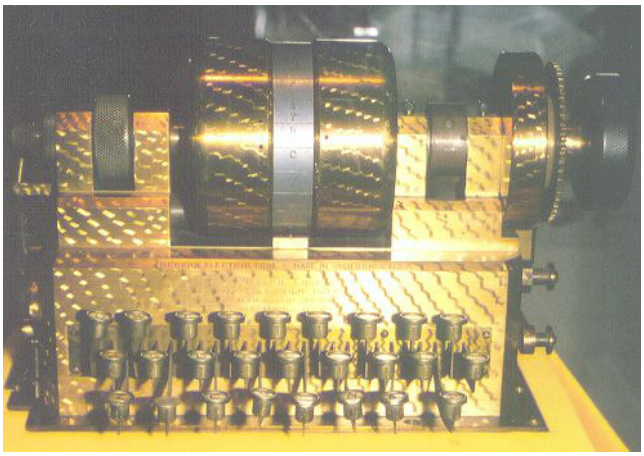


Figura 2.6 Primera máquina de código eléctrico de Hebern

2.2.1.6 El convertidor M-209 (CSP-1500) de Hagelin

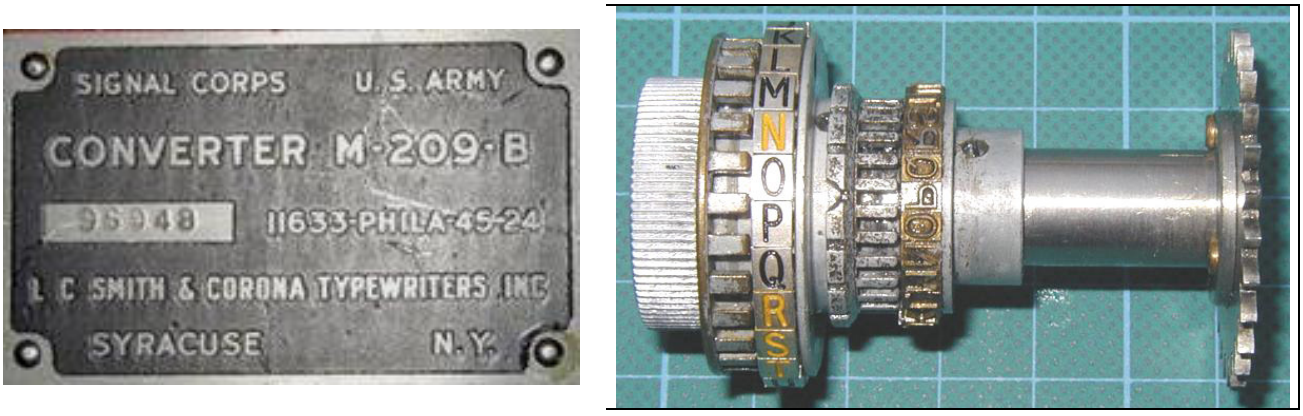


Figura 2.7 Placa y rotor del convertidor M-209

Fabricado por Boris Hagelin a principios de los años 40 para el ejército estadounidense, el M-209 era un aparato mecánico simple que medía 18 cm x 14 cm x 9 cm. Podía ser transportado en un bolso de tela. Sus componentes principales incluían seis ruedas codificadoras de 26, 25, 23, 21, 19 y 17 posiciones respectivamente, asegurando una longitud de ciclo (es decir, el número de etapas antes de que una clave dada se repita) de 101,405,850 pasos.

Este aparato fue popular debido a que era pequeño, liviano (6 libras) y el entrenamiento para utilizarlo se impartía en unas pocas horas. Para manipular este aparato, el usuario debía girar un

botón externo (situado a la izquierda) a fin de seleccionar la letra del texto claro, luego girar la manivela situada a la derecha para activar los componentes internos. Al final de este giro, el aparato imprimía la letra cifrada sobre una tira de papel. Durante la Segunda Guerra Mundial, más de 140,000 unidades de este aparato fueron fabricadas principalmente por la empresa "Smith Corona Typewriter" de los Estados Unidos.

2.2.1.7 La máquina Khrya

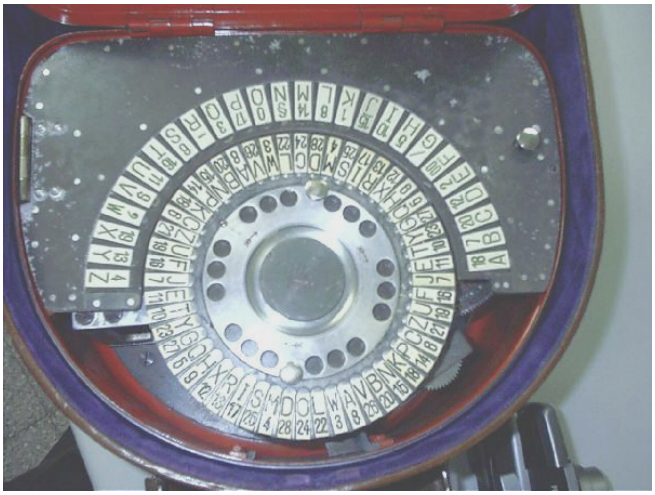


Figura 2.8 Máquina Khyra estándar

Esta máquina criptográfica apareció en 1924, su inventor fue el ucraniano Alexander von Khrya. Este aparato tenía un semicírculo fijo de letras contra el cual estaba yuxtapuesto un disco codificador dotado de engranajes que controlaban el número de desfases del disco. Una manivela servía para volver a montar un potente resorte que arrastraba la plataforma sobre la cual estaba instalado el disco codificador interior. La clave consistía en otra rueda cuyos segmentos (abiertos o cerrados) controlaban la rotación del disco codificador interior.

A pesar de su bonita apariencia y su pequeño tamaño, este aparato ejecutaba esencialmente de una sustitución con un ciclo de cifrado, cuyo período no era otro que el de algunos cientos de caracteres. Criptoanalistas estadounidenses demostraron que este aparato era susceptible de ser vulnerado en pocas horas. En 1933 Friedman descifró un criptograma de 1,135 caracteres de la versión estándar de Khyra en 2 horas y 41 minutos.

Existieron tres modelos de ésta máquina: la estándar, la versión de bolsillo conocida como "Liliput" y la Kryra electrónica.

2.2.1.8 Las máquinas TYPEX y SIGABA

Después de un largo estudio (1926-1935) sobre máquinas de cifrado comerciales como las de Hebern, Khrya y ENIGMA, un comité interministerial británico adoptó una máquina semejante a la ENIGMA llamada TYPEX. El modelo Mark III de TYPEX tenía 5 ruedas codificadoras intercambiables dotadas de un movimiento irregular. Este aparato eléctrico era pesado e imprimía el texto cifrado en tiras de papel, el impresor estaba situado en la parte posterior de la máquina. Varias versiones de la TYPEX fueron empleadas por el ejército británico y la Real Fuerza Aérea británica. Canadá también las empleó, sobre todo en el seno del Ministerio de Defensa.

Justo antes de 1940, el ejército y la marina de los Estados Unidos adoptaron una máquina similar llamada ECM Mark II. ECM significaba *Electric Cipher Machine*. El ejército estadounidense le dio el nombre de SIGABA. Solo los estadounidenses utilizaron esta máquina, especialmente durante la Segunda Guerra Mundial y un período posterior.

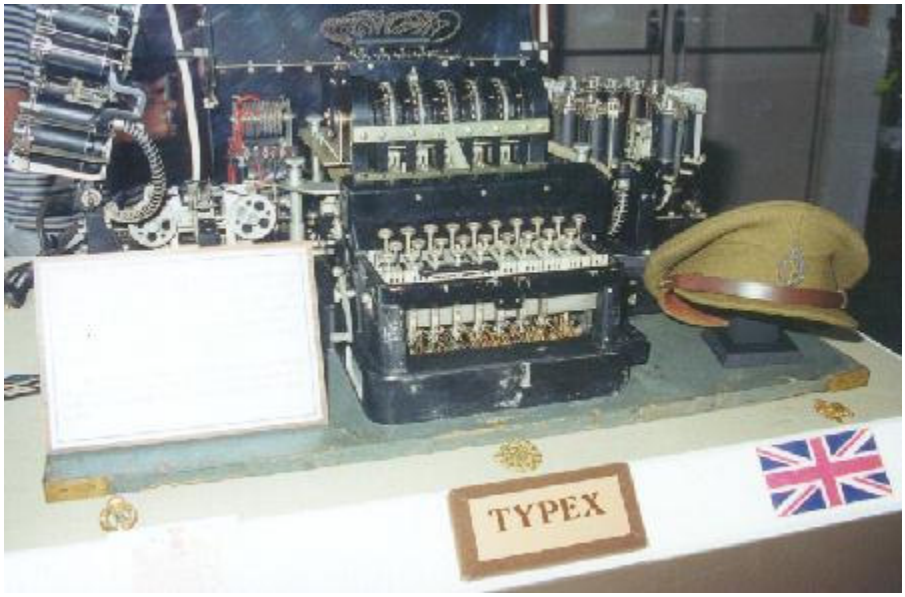


Figura 2.9 La máquina TYPEX

A fin de poder intercambiar mensajes cifrados entre la TYPEX y la ECM, se concibió para esta última un conjunto especial de ruedas codificadoras compatible con la TYPEX. Equipada de esta forma, la ECM pasó a ser llamada CCM, que significaba *Combined Cipher Machine*. Hasta donde se sabe, ningún aparato similar fue fabricado para volver a la TYPEX compatible con la ECM.

2.2.1.9 La máquina NEMA

La NEMA (*NEue MAchine*) fue puesta en servicio en 1947 por la empresa Zellwager A.G. en Suiza. Su concepción dio inicio en 1941 y dos prototipos habían sido fabricados en 1944 por cuenta de la Oficina de Cifrado del Ejército Suizo.

La NEMA compartía varias características de ENIGMA, tales como un reflector y 10 ruedas codificadoras. Por el contrario, la NEMA permitía un movimiento bastante más irregular de las ruedas codificadoras, así como del reflector. 5 ruedas codificadoras, llamadas *Fortschaltwalzen* controlaban el movimiento de las otras cinco. La situada más a la derecha era roja, mientras que las nueve restantes eran negras.

La NEMA tenía una alimentación externa ajustable (110/220 V), un adaptador para la conexión de un bombillo y un tablero luminoso inamovible que facilitaba el empleo. Los primeros usuarios de esta máquina fueron el ejército y el servicio diplomático suizo. Ambos modelos eran identificados por el color de sus ruedas.

Figura 2.10 La máquina NEMA

2.2.2 Caracterización de la era técnica

Hasta la Segunda Guerra Mundial, el rol de la criptografía, fortuito o no, fue decepcionante desde una perspectiva teórica, sin embargo jugó un papel importante en el nacimiento de la informática[38] y sería injusto negar todo status teórico a sus contribuciones: un enfoque abstracto del cifrado convencional fue en efecto iniciado por Shannon en su artículo de 1949 titulado "Communication Theory of Secrecy Systems"[39] y, precisamente trabajos como los de Shannon, habrían de desencadenar el advenimiento de la siguiente era de la criptografía. Por otra par6te, algunas décadas antes de los trabajos de Shannon, Vernam ya había dado con una forma de garantizar la confidencialidad de un proceso criptográfico, si bien las condiciones requeridas eran igualmente difíciles de satisfacer.

2.3 Era digital

Esta era está fuertemente marcada por un alto grado de teorización. Son especialmente notorias las contribuciones de Shannon y su teoría de la información, la teoría de la complejidad, la formulación de las nociones de *zero-knowledge* (o *no-divulgación*) y clave pública. A continuación se procede a describir en qué consisten estos aportes.

2.3.1 Los aportes de Shannon

Con las investigaciones de Shannon se abren las puertas a un desarrollo sin precedentes en materia de telecomunicaciones, informática y criptografía. Sus trabajos no se limitaron al

[38] Tal como indica Jean-Louis Poss, el criptoanálisis de la ENIGMA fue realizado por los polacos y luego, sobre todo, por los británicos. Bajo la dirección de Alan Turing (a quien se considera "Padre de la teoría de la computación"), el gobierno británico agrupó en Bletchley Park a destacados universitarios: matemáticos, historiadores, lingüistas, y toda una variedad de científicos y técnicos. A fines del conflicto, cerca de 7,000 personas trabajaban para la Operación ULTRA, y fue precisamente para criptoanalizar la ENIGMA que se crearon las COLOSSUS, que junto a la ENIAC son reconocidas como las primeras computadoras. Poss, J. *Introduction à la cryptographie*. École Nationale d'Arts et Métiers. Aix-en-Provence, France. Juin 2003. pp. 7.

[39] Stern, J. Op. Cit. pp. 73.

establecimiento de las bases de la teoría de la información. Precisamente a la luz de ésta, llevó a cabo investigaciones que le permitieron teorizar sobre las condiciones ideales que debía poseer un sistema criptográfico. En 1949 haría público material que se considera un hito en la historia de la criptografía[40].

En este material, recurría a análisis probabilísticos, imaginando el mensaje claro y la llave como elementos tomados aleatoriamente. Distingue la distribución de probabilidades *a priori* de la distribución *a posteriori*, luego de la intercepción de un criptograma, y hace de esta diferencia la única fuente de información del criptoanalista. Para poner un ejemplo simple, considérese que una palabra de tres letras ha sido cifrada por medio de una sustitución simple y que se sabe que dicha palabra pertenece a la lista *vos*, *del*, *que*, *ala*. Al carecer de información adicional, puede suponerse que cada uno de los elementos de esta lista tiene una probabilidad de 0.25 de representar la palabra del mensaje claro. Si luego se intercepta el criptograma, por decir algo la secuencia de letras NMS, la palabra *ala* puede descartarse de la lista de posibilidades, pues su primera y su última letra no pueden estar cifradas de forma distinta (partiendo de que se dijo que se trataba de una sustitución monoalfabética). Esta palabra tiene, entonces, una probabilidad a posteriori nula, en tanto que las otras palabras de la lista incrementan su probabilidad a 0.33. Shannon sostenía que un sistema ofrece una confidencialidad perfecta cuando, en términos actuales, es *incondicionalmente seguro*, si las distribuciones de probabilidad de los símbolos a priori y a posteriori son idénticas. Shannon explica que esto es posible, pero que implica que el número de claves deba ser al menos igual al de mensajes. En otros términos, el teorema de Shannon reclama una clave tan larga como el mensaje a cifrar. Tal como se señaló en apartados anteriores, una forma de cifrado con características tales había sido formulada alrededor de los años 20 por el ingeniero estadounidense Gilbert Vernam, de quien la idea aun conserva el nombre.

Para otros sistemas, provistos de claves fijas y sustancialmente más cortas que el mensaje, Shannon demuestra que la modificación de la distribución de probabilidades observada en el ejemplo antes presentado se agrava a medida que fragmentos más grandes de un criptograma son interceptados. Progresivamente se puede aislar un único mensaje cuya probabilidad a posteriori es extremadamente próxima a 1, mientras que el resto tiene una probabilidad despreciable; es así que el cifrado por sustitución simple de un fragmento de una frase escrito en lengua natural que contenga solamente unas treinta letras basta –teóricamente- para definir el texto claro de manera única. Esta es la expresión matemática de una experiencia que ya era familiar para los criptólogos de la era artesanal. Hay que reconocer que los conocimientos del siglo XIX no permitían discriminar mucho lo que es materialmente indescifrable en la práctica, de lo que es absoluta o matemáticamente seguro.

Es asombroso ver que Shannon estaba en capacidad de dar una definición simple desde el punto de vista de unicidad, es decir, de la longitud mínima del criptograma a partir de la cual el fenómeno aparece. Para ello recurre a la noción de *entropía*[41], que había postulado en su estudio matemático anterior sobre los sistemas de comunicación y que esta a la base de lo que ahora conocemos como teoría de la información: cada mensaje *m* que tiene una probabilidad de

[40] Shannon, C. *Communication Theory of Secrecy Systems*. Bell System Technical Journal. 1949. pp. 656-715. El contenido de este artículo formó parte de un reporte confidencial titulado "A Mathematical Theory of Cryptography", fechado el 1 de septiembre de 1946, que recientemente ha sido desclasificado.

[41] Se considera la entropía como una medida del desorden presente en el sistema. El diccionario de la lengua española lo define como:"medida de la incertidumbre existente ante un conjunto de mensajes, del cual va a recibirse uno solo".

aparición *P(m)* contribuye a la entropía en una cantidad $-P(m)\log_2(P(m))$; la entropía de una distribución es la suma de sus contribuciones.

El resultado de los trabajos de las investigaciones antes mencionadas puede interpretarse hasta cierto punto como decepcionante, ya que enunciaba que la única manera segura de cifrar era emplear una clave que tenga aproximadamente las mismas dimensiones que el mensaje claro. Faltaba a Shannon un concepto que permitiera distinguir lo que es absolutamente seguro de lo que es seguro en la práctica, que solo podía abrir el camino a las paradojas de la criptografía moderna.

Se ve aparecer, entonces, la limitación fundamental inherente al enfoque de Shannon: su teoría no logra medir más que la seguridad de un sistema criptográfico ante un adversario que dispone, siguiendo sus propios términos, de tiempo y mano de obra ilimitada. Pero en la realidad, ¿quién dispone de tales condiciones? El costo de un sistema a lo Vernam resulta prohibitivo en términos de generación y gestión de claves. La cuestión es entonces saber lo que se requiere invertir para estar protegido de máquinas imperfectas, y proveerse de una teoría capaz de informar de ello.

A este punto, es importante recordar que la base de los trabajos de Shannon, así como de otros importantes investigadores de su época, se dieron al servicio de fines militares, como había venido siendo habitual desde hacía siglos. El paso del ámbito militar al civil no se dio sin consecuencias. Progresivamente más sectores de la sociedad (sobre todo el comercio) fueron demandando servicios criptográficos y ciertas funciones fueron siendo requeridas sistemáticamente: se puede desear solamente garantizar la identidad de un interlocutor o un documento, o más simplemente la protección de un archivo contra toda modificación de parte de terceros. Se ve aparecer así la trilogía fundamental de la criptografía moderna: *confidencialidad*, *autenticidad* e *integridad*. Si bien puede considerarse que tales características podían haber sido deseables desde épocas anteriores, es hasta en la última mitad del siglo XX que esta noción se incorpora como parte de las teorías que dan soporte a la criptografía[42].

2.3.2 Surgimiento de los conceptos de complejidad, clave pública y zero-knowledge

La realización de ciertas posibilidades como las anteriormente evocadas supone aparentemente una paradoja: tratándose de la integridad de un archivo de datos informáticos, una solución natural consiste en asociar a este archivo (cuyo tamaño puede ser considerable) una huella formada por una pequeña secuencia de ceros y unos (cien o doscientos a lo sumo), y conservar o transmitir esta huella por separado. Para impedir toda modificación es necesario que un adversario no pueda presentar una versión distinta del archivo inicial con la misma huella. Desde un punto de vista matemático, esta solución es simple y llanamente complicada. Resta encontrar una teoría en la cual la construcción considerada cobre sentido: esta fue desarrollada a fines de los años 60 para brindar un marco riguroso a la noción de complejidad algorítmica.

Un algoritmo es la indicación de operaciones a efectuar mecánicamente sobre una serie de datos, y su complejidad mide la relación (generalmente creciente) entre el número de operaciones que debe efectuar la máquina para llevar a término los cálculos y el volumen de los datos. La formulación precisa de esta medida es de naturaleza asintótica, toma en cuenta lo que ocurre cuando el tamaño de los datos se vuelve infinitamente grande. En todo caso, existen algoritmos eficaces cuya complejidad se torna rápidamente grande. Tratándose de funciones que aseguran la integridad, se requiere que todo algoritmo que permita encontrar *colisiones* (es decir, archivos diferentes pero con huellas idénticas) sea impracticable. Incluso hace falta que el cálculo de la

[42] Stern, J. Op. Cit. 73-84.

huella sea impracticable. Surge así bajo una forma plausible la asimetría fundamental deseada por los criptólogos: que el cifrado y el descifrado sean sencillos y fáciles, pero el decriptamiento este totalmente fuera de alcance. Basta entonces, para escapar a la previsión de Shannon, renunciar al absoluto matemático para sustituirlo por un concepto de seguridad algorítmica. En esto consiste esencialmente la contribución de la informática a la criptografía.

La contribución antes mencionada conduce directamente a la paradoja de los llamados *sistemas de clave pública*: nada, en el marco de la criptografía ideal que sea evocado antes, exige la necesidad de una clave secreta para cifrar. Sólo el decriptamiento debe ser inaccesible al adversario. Nada prohíbe, entonces, considerar desde un punto de vista informático la existencia de una función cuyo algoritmo de cálculo sea eficaz sin que pueda ser invertido por ningún algoritmo practicable, salvo para quien disponga de cierta información suplementaria, que es precisamente la clave secreta de descifrado. A tales funciones se da el nombre genérico de *funciones de sentido único*.

Para evaluar en qué medida la criptología accede al rango de ciencia, hay que retomar el análisis de una de las funciones conferidas a la criptografía moderna: la autenticidad, que a su vez lleva la creación del concepto *zero-knowledge*. La cuestión acá es la identificación, procedimiento por el cual una entidad prueba su identidad a otra. La solución más simple a este problema es la tradicional contraseña; la imagen para una función de sentido único de esta contraseña es conocida por el dispositivo de control y basta presentarla o revelarla para identificarse. Son obvias las dificultades que presenta un sistema de este tipo, a pesar de lo cual aún es ampliamente utilizado. El *zero-knowledge* permite probar que se conoce la contraseña sin que se requiera revelar la menor parte de la misma; más exactamente, se procede a un intercambio de preguntas y respuestas entre la entidad a identificar y el dispositivo de control, de manera que para el proceso el dispositivo de control se convenza de la entidad de su interlocutor, de tal forma que la información intercambiada durante el protocolo no difiera estadísticamente de una secuencia de unos y ceros que habría sido generada por un programa que no tiene acceso al secreto. El éxito de esta idea reside en su aspecto paradójico como en las posibilidades reales de su aplicación.

El año 1976 marca una ruptura decisiva en la criptografía. En su historia surge un "antes" y un "después" a partir del concepto de clave pública presentado por Whitfield Diffie y Martin Hellman[43].

La importancia de su planteamiento no escapó a sus autores quienes en su artículo, que habría de convertirse en el más citado de la literatura sobre criptografía, escriben desde sus primeras líneas: "Nos encontramos hoy día a la alba de una revolución en la criptografía"[44]. Una revolución es precisamente de lo que trata, con el potenciamiento brutal de recursos al servicio de la criptografía y más ampliamente en el campo del procesamiento de la información se lograron

[43] Diffie, W., Hellman, M. *New directions in criptography*, manuscrito recibido el 31 de junio de 1976, trabajo que fue parcialmente apoyado por la NSF (National Science Fooundation). Partes del documento fueron presentadas al IEEE en su Taller de teoría de la información, Lenox, MA, 23-25 de junio, 1975 y el Simposium internacional de la teoría de la información en Suecia, 21-24 de junio, 1976.

[44] "We stand today on the brink of a revolution in cryptography. The development of cheap digital hardware has freed it from the design limitations of mechanical computing and brought the cost of high grade cryptographic devices down to where they can be used in such commercial applications as remote cash dispensers and computer terminals. In turn, such applications create a need for new types of cryptographic systems which minimize the necessity of secure key distribution channels and supply the equivalent of a written signature. At the same time, theoretical developments in information theory and computer science show promise of providing provably secure cryptosystems, changing this ancient art into a science." Diffie, W., Hellman, M. Op. Cit. pp. 1.

mecanismos hasta entonces no imaginables. El volumen de datos que circula sin cesar por el mundo vuelve impracticable la utilización sistemática de canales *asegurados,* y es entonces la información misma la que debe ser protegida. Para aclarar esta idea, considérese el origen de la invención de la clave pública.

Una de las cuestiones a la base de esta invención es la necesidad de resolver de forma práctica el problema de distribución de claves. No se trata más que de una versión particular de un problema al que todo mundo se ve confrontado cotidianamente, la acumulación: la llave de la casa, la del carro, la de la oficina, la de una residencia secundaria, etc. Tal como en la vida cotidiana, la solución de una clave única no es razonable: resulta conveniente que hayan claves para distintos niveles de acceso (llaves maestras, llaves de operador, llaves de usuario, llaves de sesión, etc.). Actualmente estas claves en general ya no son palabras o frases , sino simplemente arreglos de cifras, algo así como los números de teléfono pero notablemente más largos. Es necesario poder transmitirlos de forma segura a su usuario legítimo, y poder salvaguardarlos de manera confidencial.

Ante este problema, la solución propuesta por Diffie-Hellman es sumamente sencilla: para suprimir los problemas de transferencia y almacenamiento de llaves puede publicárselas en un diario o un anuario. ¡Puede decirse que entonces las claves ya no son secretas!, lo cual es cierto, pero ¿porqué habrían de ser simétricas las operaciones de cifrado y descifrado? y ¿porqué la clave que permite a un usuario crear un mensaje codificado habría de ser idéntica a la que usa el destinatario para recuperar el mensaje claro? Estas condiciones no son imprescindibles y su persistencia se ha debido más bien a prácticas establecidas por siglos de criptología convencional, y por una visión excesiva de la seguridad que pretendería impedir al enemigo cifrar cuando basta solamente no permitirle decriptar.

La gran audacia de Diffie-Hellman consistió precisamente en proponer el concepto de cifrado asimétrico, en el cual los mecanismos de codificación y decodificación son distintos. Una clave pública autoriza la primera operación, mientras que la segunda necesita una clave distinta y secreta.

Para enfatizar la actualidad de esta idea nótese que el concepto de clave pública fue descrito en términos que casi 30 años más tarde son aún los que utilizan los manuales de criptografía. La simple realización práctica de la idea propuesta por el artículo es un mecanismo llamado intercambio público de llaves, que permite realizar una funcionalidad más débil que la clave pública. Pese a esta laguna es innegable que la historia dio la razón a Diffie-Hellman, cuya fe en la pertinencia de sus ideas era tal que no pudieron retrazar la publicación de su descubrimiento, permitiendo a otros la tarea de proponer concretamente un verdadero sistema criptográfico de clave pública.

Es entonces que a Ronald Rivest, Adi Shamir y Leonard Adleman corresponde en 1978 el privilegio de presentar el primer sistema de clave pública, conocido ahora por el acrónimo RSA (iniciales de los apellidos de sus creadores). Adi Shamir relata con agrado la ironía de este descubrimiento: él se había vuelto amigo de Rivest y Adleman durante una estancia post-doctoral en el Instituto Tecnológico de Massachussets; ellos decidieron entonces unir esfuerzos para trabajar sobre el artículo de Diffie y Hellman (que acababa de ser publicado), con la idea de demostrar que el concepto de clave pública era intrínsecamente contradictorio. Tras numerosos ensayos, dieron finalmente con el resultado opuesto: concibieron un sistema criptográfico

asimétrico que parecía robusto, resultado notable cuya importancia supieron medir en seguida los tres autores, lo cual los motivó a publicarlo de inmediatamente[45].

En la misma época, Hellman y uno de sus estudiantes, Ralph Merkle, propusieron otro sistema de clave pública basado en un mecanismo diferente. Mientras el RSA utiliza la aritmética de los grandes números enteros, el sistema de Hellman y Merkle recurre al uso de un problema combinatorio conocido con el nombre de *problema de la mochila*; el esquema de Hellman y Merkle también fue publicado y patentado, con más cuidados que el RSA ya que su validez podría extenderse fuera de los Estados Unidos. Es decir que la importancia de aquello a lo que la industria le apostó, comenzaba a ser presentida por los investigadores.

La guerra de los sistemas de clave pública que podría haber resultado de la presencia de estas dos propuestas concurrentes no llegó a tener lugar, esencialmente porque el RSA se hizo rápidamente de la victoria. Este sistema presenta ventajas técnicas innegables, en particular, cifra un texto claro por un mensaje codificado del mismo tamaño, contrariamente al sistema de Hellman y Merkle, cuyo rendimiento es menor, pues la codificación alcanza o excede el doble del tamaño del texto claro. Además, el sistema de Hellman y Merkle tiene el inconveniente de producir claves públicas mucho más voluminosas del RSA. Pero aún hay algo más grave: la seguridad del sistema de Hellman y Merkle siempre pareció sospechosa a numerosos expertos; esta intuición se vio confirmada cuando en 1982, durante el congreso Crypto[46], Adi Shamir presentó un ataque contra los sistemas criptográficos basados en el problema de la mochila. Para volver aún más dramático el efecto anunciado Shamir no dudó en colocar sobre el estrado una computadora personal e ilustrar mediante experimentos la validez de su ataque, mostrando que autorizaba la recuperación de un texto claro a partir solamente del texto cifrado y la clave pública.

Existe una ambigüedad fundamental en cuanto al rol de las matemáticas en los sistemas criptográficos asimétricos. Ofrecen el medio para concebir estos sistemas y dotarlos de seguridad altamente plausible; no dan, por lo tanto certeza absoluta (ni pretenden darla)[47].

Es pertinente resaltar que unos años antes de la aparición de la propuesta de Diffie y Hellman los sistemas de cifrado simétrico ya ocupaban un lugar importante, a tal grado que se fueron estableciendo estándares para fines comerciales. Es así como surgen algoritmos tan importantes como el DES que se mantuvo como referente por más de 30 años y el AES, que sustituyó al primero como estándar del NIST en el año 2000. Esta sustitución se dio debido a que el DES ya no respondía a ciertos requerimientos de seguridad, la evolución en el campo de la informática empezaba a hacerlo vulnerable. Para establecer al AES como nuevo estándar se convocó a un concurso. Las propuestas debían cumplir con criterios de: seguridad, rendimiento, eficiencia, facilidad de implementación y flexibilidad.

---

[45] Una patente fue depositada con posterioridad a la primera publicación (lo cual es permitido por la legislación estadounidense). Tal como en el caso de Diffie-Hellman, esta prisa por publicar estuvo ligada a la importancia de los resultados obtenidos, así como a la voluntad de inscribirlos en el marco de una investigación civil de vanguardia.

[46] Que reúne cada año a importantes especialistas en temas de criptología en el campus de la Universidad de Santa Bárbara en California.

[47] Stern, J. Op. Cit. pp. 85-90.
Poss, J. *Introduction à la cryptographie.* École Nationale d'Arts et Métiers. Aix-en-Provence, France. Juin 2003. pp. 8-9, 30.
Zaccagnini, A. *Introduzione alla crittografia*. Università degli Studi di Parma. Parma. 2002 pp. 39-40, 43.

Luego de haber evaluado las propuestas, se consideró que la formulada por los investigadores belgas Daemen y Rijmen, el algoritmo Rijndael, respondía de una forma más integral y suficientemente buena a los criterios de diseño planteados como requisito para la presentación de propuestas[48].

Los desarrollos más recientes en matemáticas aplicadas a la criptografía parecen indicar que en el futuro habrán mas sistemas basados en curvas elípticas, y computación cuántica. El presente trabajo no se ocupará de estos aspectos.

[48]Schneier, B., Whiting, D. *A Performance Comparison of the Five AES Finalists*. Counterpane Internet Security-Hi/Fn. 7 April 2000

3

Métodos modernos de criptografía digital

Los sistemas de cifrado se clasifican comúnmente en sistemas de cifrado de bloque y sistemas de cifrado de flujo. A continuación se describen las características de ambos.

3.1 Sistemas de cifrado de bloque

Son los que dividen el mensaje a cifrar en distintos fragmentos, usualmente de longitud fija, y aplican a cada uno de ellos una transformación criptográfica.

Los sistemas de cifrado de bloque se dividen a su vez en simétricos y asimétricos. Se habla de criptografía simétrica cuando se utiliza la misma clave tanto para cifrar como para descifrar. La robustez del cifrado depende, en tal caso, de la exclusividad en el conocimiento de la clave, como señala el principio de Kerckhoffs[49].

3.1.1 Sistemas de cifrado de bloque simétrico

La gran mayoría de los algoritmos de cifrado simétricos se apoyan en los conceptos de *confusión* y *difusión*, estas técnicas consisten básicamente en truncar el mensaje en bloques de tamaño fijo, y aplicar la función de cifrado a cada uno de ellos.

[49] Ya mencionado con anterioridad en este documento, el principio de Kerckhoffs postula que la seguridad de un sistema criptográfico está basada exclusivamente en el conocimiento de la clave, en la práctica se presupone conocido a priori el algoritmo de cifrado y descifrado.
Zimuel, E. Introduzione a la crittografia. Mensa Italia. Francavilla al Mare. *Luglio 2002.*

La confusión consiste en tratar de ocultar la relación que existe entre el texto claro, el texto cifrado y la clave. Un buen mecanismo de confusión hará demasiado complicado extraer relaciones estadísticas entre las tres cosas. Por su parte Difusión trata de repartir la influencia de cada bit del mensaje original lo más posible entre el mensaje cifrado.

La confusión por sí sola sería suficiente, ya que si establecemos una tabla de sustitución completamente diferente para cada clave con todos los textos claros posibles tendremos un sistema extremadamente seguro. Sin embargo, dichas tablas ocuparían cantidades astronómicas de memoria.

Para obtener algoritmos fuertes sin necesidad de almacenar tablas enormes se intercala la confusión (sustituciones simples, con tablas pequeñas) y la difusión (permutaciones). Esta combinación se conoce como *cifrado de producto*.

En muchos casos el criptosistema no es más que una operación combinada de sustituciones y permutaciones, repetida n veces.

3.1.2 Modos de cifrado de bloque

Se debe considerar el caso en que la longitud del bloque que se desea cifrar no es un múltiplo exacto del tamaño del bloque. En tal caso es necesario añadir información adicional al final para que sí lo sea. Lo más sencillo es rellenar con ceros (o algún otro patrón) el último bloque que se codifica. El inconveniente de esto se puede presentar cuando se descifra para saber donde hay que cortar el bloque. Para solventar este problema, se suele agregar como último byte del último bloque el número de bytes que se han añadido

Esto presenta desventajas sí el tamaño original es múltiplo del bloque, en tal caso hay que alargarlo con un bloque entero. Por ejemplo, sí el tamaño del bloque fuera de 64 bits y sobraran 5 bytes al final, se añadirían dos ceros y un tres, para completar los ocho bytes necesarios en el último bloque. Si por el contrario no sobrara nada habría que añadir siete ceros y un ocho, tal como se indica en la figura 3.1.

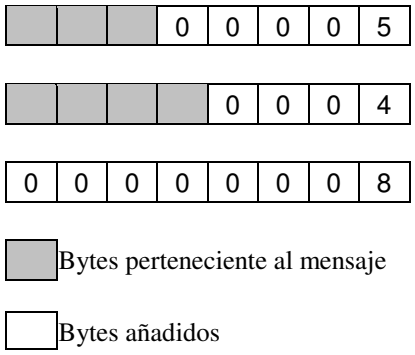


Figura 3.1 Relleno de bytes en cifrado por bloques

3.1.2.1 ECB - Electronic codebook

El método más simple para cifrar textos claros largos, el procedimiento es el siguiente: se cifra por separado cada bloque con el mensaje. Este modo de cifrado no es utilizado en la práctica pues presenta fuertes debilidades:

- Si dos bloques de texto claro son iguales, entonces, el correspondiente texto cifrado será idéntico, y esto será es peligroso, sobre todo cuando se codifica información muy redundante, o con patrones acomunes al inicio y al final. Un adversario puede someter el proceso a un ataque basado en análisis de frecuencias y extraer información;

- El atacante puede cambiar un bloque sin mayores problemas, y alterar los mensajes incluso desconociendo la clave y el algoritmo empleados.

3.1.2.2 CBC – Cipher block chaining

Es el modo de cifrado de bloques más ampliamente usado, ya que incorpora un mecanismo de retroalimentación. La codificación del primer bloque condiciona la de los siguientes, imposibilitando la sustitución de un bloque individual en el mensaje cifrado. Los problemas de ECB se evitan al aplicar la función XOR entre un bloque del mensaje a codificar y el último bloque ya cifrado.

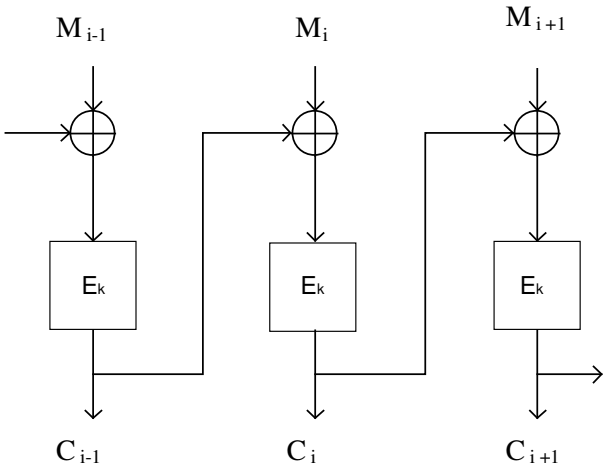


Figura 3.2 Modo de operación CBC, codificación

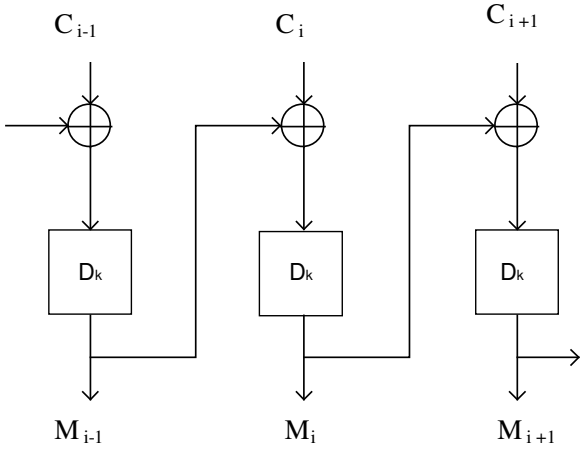


Figura 3.3 Modo de operación CBC, decodificación

Dos mensajes idénticos se cifrarán de la misma forma usando el modo CBC. Dos mensajes que empiecen igual se codificarán igual hasta llegar a la primera diferencia entre ellos. Para evitar esto se usa un vector de inicialización, que puede ser un bloque aleatorio, como bloque inicial de la transmisión. Este vector se descarta en el destino, pero garantiza que siempre los mensajes se cifren de forma distinta aunque tengan partes comunes.

3.1.2.3 CFB – Cipher-Feedback

Este modo no empieza a codificar o decodificar mientras no tiene que transmitir o recibir un bloque completo de información. Esto puede suponer inconvenientes.

El CFB permite cifrar la información en unidades inferiores al tamaño del bloque, con lo cual se aprovecha mejor la capacidad de transmisión del canal de comunicación, manteniendo además un nivel de seguridad adecuado, ya que no puede ser vulnerado a través de análisis estadístico.

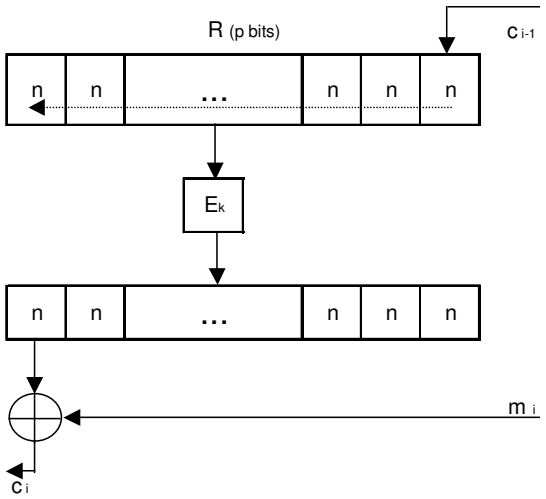


Figura 3.4 Esquema del modo de operación CFB

En la Figura 3.4 se muestra el esquema de funcionamiento de este modo de operación. Sea p el tamaño de bloque del algoritmo simétrico, y sea n el tamaño de los bloques que se desea transmitir (n debe ser divisor de p). Sea m_i el i-ésimo bloque del texto claro, de tamaño n. Se recurre entonces a un registro de desplazamiento R de longitud p al cual se carga con un vector de inicialización. Se codifica el registro con el algoritmo simétrico y se obtiene en r sus n bits más a la izquierda. El bloque a enviar es $c_i = r \; XOR \; m_i$. Se desplaza R n bits a la izquierda y se introduce c_i por la derecha.

Para descifrar basta con cargar el vector de inicialización en R y codificarlo, calculando r. Entonces, $m_i = r \; XOR \; c_i$. Se desplaza R y se introduce c_i por la derecha. Es de hacer notar que si $n = p$, el modo CFB se reduce al CBC.

3.2 Sistema de cifrado de flujo

En 1917 Joseph Mauborgne y Gilbert Vernam crearon un criptosistema que cumplía los criterios de perfección planteados por Shannon. En este sistema se emplea una secuencia aleatoria cuya longitud era igual a la del mensaje, que se usaría sólo una vez (lo que se conoce en inglés como

one time pad), combinándola mediante una función simple y reversible (usualmente un XOR) con el mensaje claro carácter a carácter.

Sin embargo, si la clave es tan larga como el propio mensaje y se dispone de un canal seguro para enviarla ¿porque no emplearse para transmitir el mensaje directamente?.

Un sistema de Vernam carece de utilidad práctica en la mayoría de casos. No obstante si se dispone de un generador pseudo-aleatorio capaz de generar secuencias "criptográficamente aleatorias", de forma que la longitud de los posibles ciclos sea extremadamente grande, sería posible, empleando la semilla del generador como clave, obtener cadenas de bits desechables, y emplearlas para cifrar mensajes aplicando simplemente la función XOR entre el texto en claro y la secuencia generada. Al conocerse la semilla, podrá reconstruirse la secuencia pseudo-aleatoria y descifrarse el mensaje.

Estos sistemas dividen el mensaje a cifrar en caracteres (de menor longitud que los bloques), cifrando cada carácter mediante una función que varia en el tiempo.

A diferencia del sistema de cifrado de bloques, en este sistema, caracteres idénticos poseen por lo general cifrados diferentes, lo que contribuye a aumentar la seguridad del sistema.

Existen dos tipos básicos de cifradores de flujo: los síncronos y los autosincronizantes, el criterio para establecer la diferencia entre ambos es el generador de las secuencias pseudo-aleatorias.

3.2.1 Sistemas de cifrado de flujo síncrono

En un sistema síncrono la secuencia es calculada de forma independiente tanto del mensaje claro como del cifrado. En el caso general, ilustrado en la figura 11.1 a, viene dado por las siguientes ecuaciones:

$$s_{i+1} = g(s_i, k)$$
$$o_i = h(s_i, k)$$
$$c_i = w(m_i, o_i)$$

Donde k es la clave, si es el estado interno del generador, s_0 es el estado inicial, o_i es la salida en el instante i, m_i y c_i son la i-ésima porción del texto claro y cifrado respectivamente, y w es una función reversible, usualmente XOR. En muchos casos, la función h depende únicamente de sí, sigue siendo $k = s_0$.

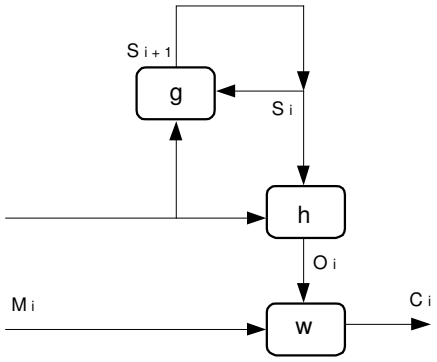


Figura 3.5 Esquema de generadores de secuencia. Generador síncrono

Cuando empleamos un generador de estas características, necesitamos que tanto el emisor como el receptor estén sincronizados para que el texto pueda descifrarse. Si durante la transmisión se pierde o inserta algún bit, ya no se estará aplicando en el receptor XOR con la misma secuencia, por lo que el resto del mensaje será imposible de descifrar. Esto nos obliga a emplear tanto técnicas de verificación como de reestablecimiento de la sincronía.

Otro problema muy común con este tipo de técnicas es que si algún bit del criptograma es alterado, la sincronización no se pierde, pero el texto claro se verá modificado en la misma posición. Esta característica podría permitir a un atacante introducir cambios en nuestros mensajes, simplemente conociendo qué bits debe alterar. Para evitar esto, deben emplearse mecanismos de verificación que garanticen la inseguridad del mensaje recibido, como las funciones *Hash*.

3.2.2 Sistemas de cifrado de flujo asíncrono o autosincronizante

Son aquellos sistemas en los que la secuencia generada es función de una semilla, más una cantidad fija de los bits anteriores de la propia secuencia.

Estos sistemas son resistentes a la pérdida o inserción de información, ya que vuelven a sincronizarse automáticamente en cuanto llegan t bloques correctos de forma consecutiva. También serán sensibles a la alteración de un mensaje, ya que sí se modifica la unidad de información c_i el receptor tendrá valores erróneos de entrada en su función h hasta que se alcance el bloque c_{i+t}, momento a partir del cual la transmisión habrá recuperado la sincronización. En cualquier caso, al igual que en los sistemas síncronos, es necesaria la introducción de mecanismos de verificación.

Una ventaja de estos sistemas radica en la dispersión de las propiedades estadísticas del mensaje claro a lo largo de todo el mensaje cifrado, pues cada elemento del mensaje influye en todo el criptograma. Lo anterior los vuelve más resistentes a ataques basados en la redundancia del mensaje claro.

3.2.3 Diferencia básica entre el cifrado de flujo síncrono y el autosincronizante

En el cifrado síncrono, para poder establecer la comunicación, el emisor y el receptor deben utilizar la misma clave y ésta debe estar sincronizada. Por lo tanto, este tipo de sistemas necesita señales de sincronización sin las cuales el descifrado no es posible.

En los sistemas de cifrado autosincronizante no necesitan señales de sincronización, ya que en caso de pérdida de sincronismo, éste se recupera transcurrido un cierto tiempo gracias a la retroalimentación.

Ambos métodos tienen sus ventajas e inconvenientes, los métodos síncronos son inmunes a los ataques activos de inserción de mensajes extraños, puesto que destruyen la sincronización y por lo tanto son detectables. Los métodos autosincronizantes están expuestos a ataques activos de repetición de mensajes. Estos ataques pueden evitarse utilizando mensajes adicionales de identificación previamente acordados por los comunicantes.

3.3 Modos de cifrado de flujo

3.3.1.1 OFB – Output feedback

Este modo es diferente a todos los antes mencionados, ya que el mensaje nunca es usado como entrada al bloque cifrado. En su lugar, el bloque cifrado se usa para generar un flujo seudo-aleatorio de bytes (*key stream*), al cual se aplica una función XOR con el texto claro para producir texto cifrado.

Uno de los inconvenientes de este modo es que si un eventual atacante conoce un bloque de texto claro y su correspondiente cifrado, entonces es capaz de conseguir que el receptor recupere un falso mensaje.

Una ventaja del OFB es que el proceso de descifrado usa exactamente la misma operación que la de cifrado, lo cual ahorra esfuerzos de implementación.

Una segunda ventaja es que no necesita rellenos. Si se considera al key stream como una secuencia de bytes, puede usarse tantos bytes de longitud del mensaje. En otras palabras, si el ultimo bloque de texto claro está parcialmente lleno, basta enviar los bytes del texto cifrado que corresponden a los bytes de texto claro actuales.

La gran desventaja del OFB es el riesgo de repetir el valor de la clave del bloque, después de la cual la secuencia de los bloques de clave simplemente se repiten.

3.3.1.2 CTR – Counter

Este modo usa secuencias de números como entrada a un algoritmo. En lugar de usar la salida del algoritmo de cifrado para rellenar un registro, la entrada para éste es un contador, donde, después de cada bloque de cifrado, el contador se incrementa en una constante, usualmente uno. Este modo resuelve el problema del modo OFB donde la salida es de n bits, donde n es menor que la longitud del bloque.

Modos de cifrado	
ECB	**CBC**
Seguridad	**Seguridad**
- Los patrones del texto claro no son ocultados - La entrada del cifrador por bloques no es aleatoria; es la del texto claro + Más de un mensaje puede ser cifrado con la misma llave - El texto claro es fácil de manipular; algunos bloques pueden ser eliminados, repetidos o intercambiados.	+ Los patrones de texto claro son ocultados aplicando un XOR a los bloques previos de texto cifrado + La entrada del cifrador por bloques es aleatoria al aplicar XOR con el previo bloque de texto cifrado + Más de un mensaje puede ser cifrado con la misma llave +/- El texto claro es de alguna manera dificil, de manipular; los bloques pueden ser removidos desde el inicio y final del mensaje, los bits del primer bloque pueden ser cambiados, y la repetición permite algunos cambios controlados.
Eficiencia	**Eficiencia**
+ La velocidad es la misma que la de un cifrado por bloques - El texto cifrado puede ser de hasta un bloque más largo que el texto claro debido al relleno - No es posible el procesamiento previo + El procesamiento es paralelizable	+ La velocidad es la misma que la de un cifrado por bloques - El texto cifrado puede ser hasta de un bloque más largo que el texto claro, sin contar el IV. - No es posible el procesamiento previo +/- El cifrado no es paralelizable; el descifrado es paralelizable y tiene una propiedad de acceso aleatorio.
Tolerancia a fallas	**Tolerancia a fallas**
- Un error de texto cifrado afecta un bloque completo de texto claro - Un error de sincronización es irrecuperable	- Un error de texto cifrado afecta un bloque completo de texto claro y el correspondiente bit en el bloque siguiente - Un error de sincronización es irrecuperable

Tabla 3.1 Comparación de los modos de cifrado

Modos de cifrado	
CFB	**OFB**
Seguridad	**Seguridad**
+ Los patrones de texto claro son ocultados + La entrada a un bloque de cifrado es aleatoria + Puede cifrarse más de un mensaje con la misma clave +/- El texto claro es de alguna forma difícil de manipular; los bloques del inicio y final del mensaje pueden ser removidos + Los bits del primer bloque pueden ser cambiados y la repetición permite algunos cambios controlados	+ Los patrones de texto claro están ocultos + La entrada al cifrador de bloques es aleatoria + Más de un mensaje puede ser cifrado con la misma clave - El texto claro es fácil de manipular; cualquier cambio en el texto cifrado afecta directamente al texto claro
Eficiencia	**Eficiencia**
+ La velocidad es la misma que la de cifrador de bloques - El texto cifrado es del mismo tamaño que el texto claro +/- El cifrado no es paralelizable; el descifrado es paralelizable y tiene propiedades de acceso aleatorio - Algún tipo de pre-procesamiento es posible antes de que el bloque sea visto; el bloque previo de texto cifrado puede ser cifrado	+ La velocidad es la misma que la de cifrador de bloques - El texto cifrado es del mismo tamaño que el texto claro + El procesamiento es posible antes que el mensaje sea visto +/- El procesamiento no es paralelizable
Tolerancia a fallas	**Tolerancia a fallos**
- Un error en el texto cifrado afecta al bit correspondiente del texto claro y al siguiente bloque completo + Errores de sincronización de bloque completo son recuperables	+ Un error en el texto cifrado afecta solo al bit correspondiente del texto claro - Errores de sincronización son irrecuperables

Tabla 3.1 Comparación de los modos de cifrado

4

Algoritmos de llave privada

En este capítulo se estudian varios sistemas de cifrado de llave privada. Se detalla su estructura y funcionamiento. Como paso preliminar se procede a describir algunos elementos estructurales cuya utilización es común de algunos algoritmos de cifrado.

4.1 Elementos estructurales de uso frecuente

Los elementos que se describen a continuación datan de época en que surgieron los primeros estándares de cifrado digital de bloques y se sigue recurriendo a ellos como parte de varios algoritmos contemporáneos.

4.1.1 Redes de Feistel

Una red de Feistel es un método general de transformación de cualquier función (usualmente llamada función F) en una permutación. Este procedimiento fue inventado por Horst Feistel en su diseño de Lucifer y ha sido usado en muchos diseños de algoritmo de cifrado, por ejemplo DES, FEAL, GOST, Khufu y Khafre, LOKI, CAST, Blowfish y RC5.

Es importante notar que casi todos los algoritmos de cifrado propuestos que están basados en las redes de Feistel siguen el mismo diseño: las mitades L y R (izquierda y derecha respectivamente) del bloque de bits operan sobre la otra mitad.

La construcción fundamental de las redes de Feistel es la función F: una llave dependiente de una entrada de datos que genera una salida. La función F no es lineal y casi siempre es irreversible.

El procesamiento de los bloques de bits en las redes de Feistel puede ser representada mediante la Figura siguiente:

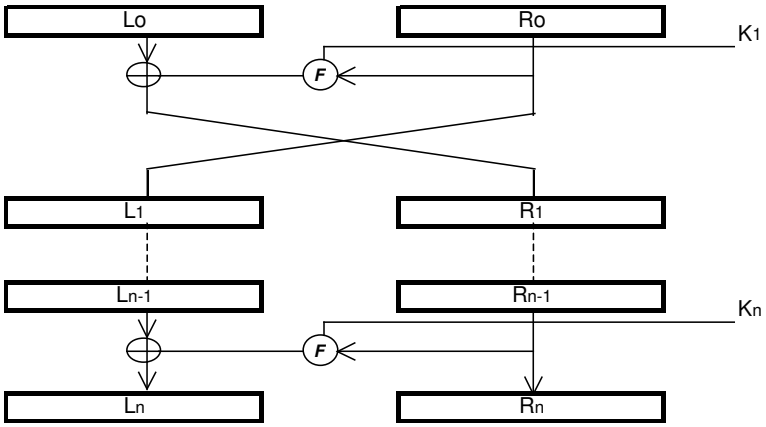


Figura 4.1 Estructura de una red de Feistel

En una red de Feistel la salida de cada vuelta se usa como entrada para la siguiente según la relación:

$$\left.\begin{aligned} Li &= R_{i-1} \\ Ri &= L_{i-1}\ \text{XOR}\ f(R_{i-1}, Ki) \end{aligned}\right\} \quad \text{si } i < n.$$

$$\begin{aligned} Ln &= L_{n-1}\ \text{XOR}\ f(R_{n-1}, K_n) \\ R_n &= R_{n-1} \end{aligned}$$

Este tipo de estructura tiene la propiedad de ser reversible, independientemente de como sea la función *F*, para ello basta aplicar de nuevo el algoritmo al resultado, pero empleando las sub-llaves K_i en orden inverso. Esto permite emplear el mismo mecanismo tanto para cifrar como para descifrar.

4.1.2 Cajas de sustitución

Las cajas de sustitución intercalan sustituciones sencillas (confusión) con tablas pequeñas y permutaciones (difusión).

Una caja de sustitución tiene mxn bits toma como entrada cadenas de m bits y da como salida cadenas de n bits. Esto se ilustra en la Figura 2. Por ejemplo, el DES emplea ocho cajas de sustitución de 6x4 bits.

La utilización de las cajas de sustitución es sencilla: Dividen el bloque original en segmentos de m bits y cada uno de ellos se sustituye por otro de n bits haciendo uso de la caja de sustitución correspondiente.

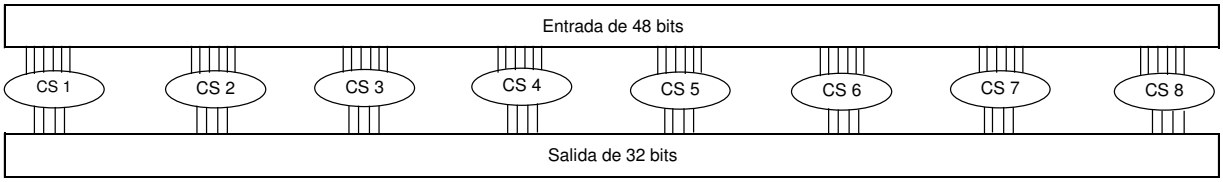


Figura 4.2 Cajas de sustitución

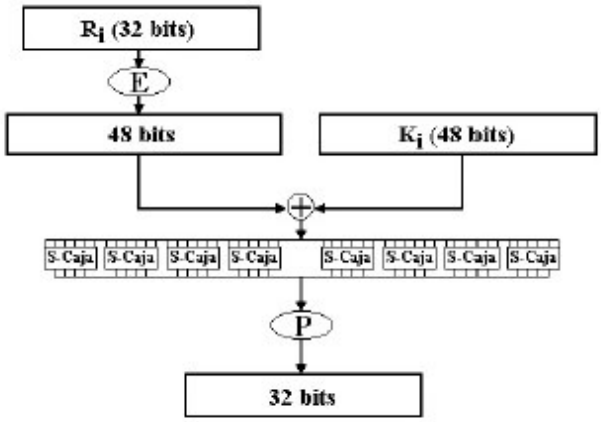


Figura 4.3 La función F y las cajas de sustitución del DES

Normalmente, mientras más grandes sean las cajas de sustitución, más resistente será el algoritmo resultante.

4.2 Lucifer

A finales de los años 70s, Horst Feistel y Walt Tuchman de IBM, iniciaron Lucifer, un programa de investigación sobre mecanismos de cifrado para computadoras. El mismo nombre fue asignado al algoritmo que obtuvieron como resultado sus investigaciones.

Lucifer, el antecesor del DES[50], es una red de sustitución-permutación de 16 vueltas, bloques de 128 bits y una generación de llaves similar a la del DES.

Al igual que el DES, Lucifer es un cifrador iterativo, que utilizaba redes de Feistel. Es decir que mezcla bloques de datos al llevar a cabo diferentes pasos de cifrado sobre los mismos bloques; este proceso involucra la llave y la mitad del bloque para calcular una salida a la cual se le aplicaba un XOR junto a la otra mitad del bloque. Entonces, las mitades del bloque son intercambiadas, de manera que ambas mitades del bloque fueran modificadas igual número de veces.
Cada vuelta utiliza una sub-llave de 72 bits. La sub-llave para la primera vuelta consiste en el primer byte de la llave repetido dos veces, seguida de los siguientes 7 bytes de la llave. Se rotan los 7 bytes izquierdos de la llave, y se genera la sub-llave para la siguiente ronda.

[50] DES: Data Encryption Standard. Más adelante se dedican varios apartados a su descripción.

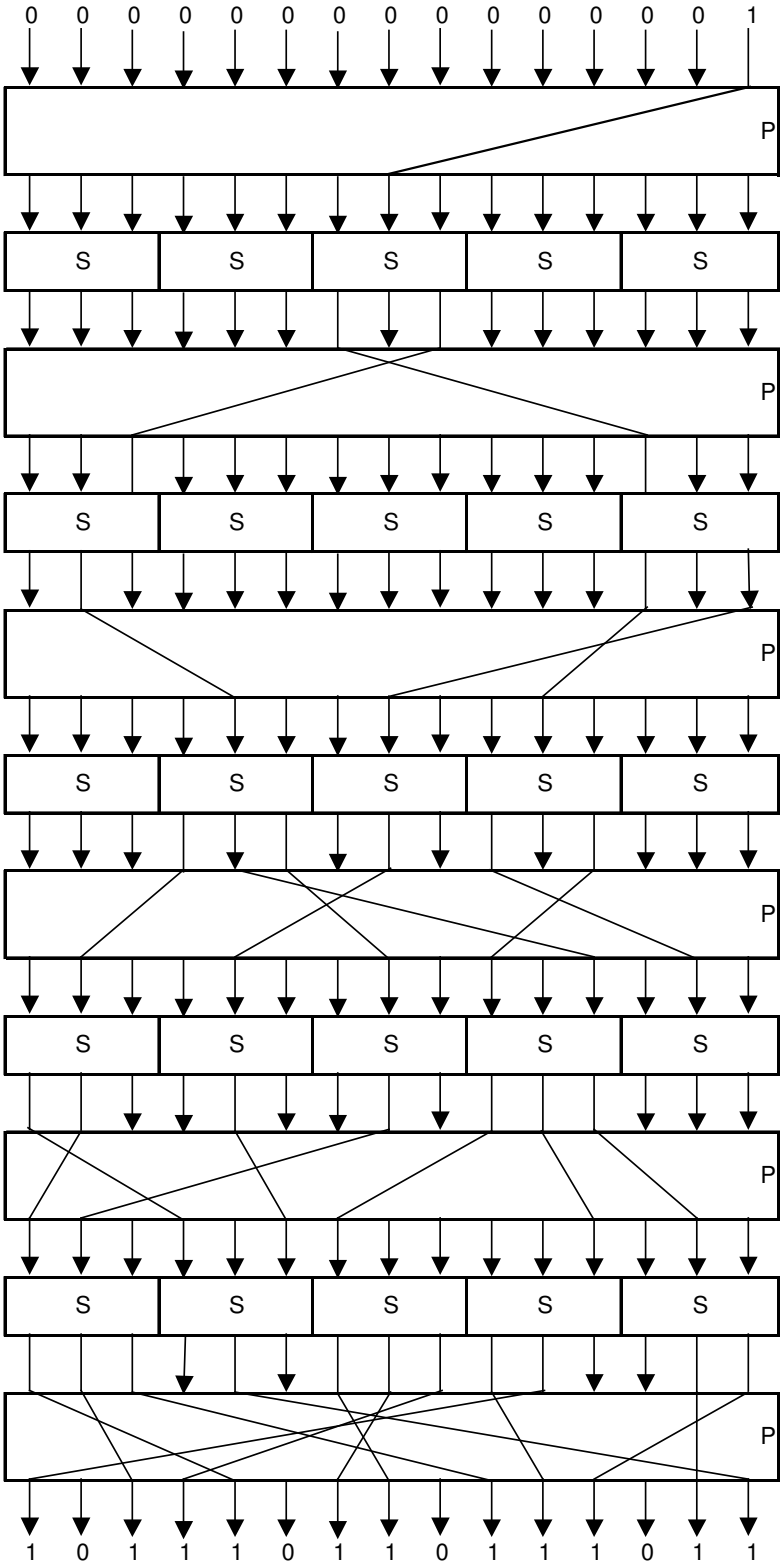


Figura 4.4 Proceso de cifrado de Lucifer

4.2.1 La función F

La función F opera sobre los 64 bits de la mitad derecha de los datos, aplicando la función XOR a dicha mitad derecha del bloque con los últimos 8 bytes de la sub-llave para cada vuelta.

Basados en los bits del primer byte de la sub-llave para esa vuelta, se intercambian los ocho bytes de dicho resultado para aquellos bits que le corresponden al bit 1.

4.2.2 Cajas de sustitución y transposición

Las cajas de sustitución (S-boxes) de Lucifer tienen entradas y salidas de 4 bits. La entrada de las cajas de sustitución es el bit permutado de caja en la vuelta anterior. La entrada de la primera caja de sustitución es el texto claro. Se usa una de bits para elegir una de entre dos cajas de sustitución posibles (S_0 y S_1). Lucifer representa lo anterior como una sola caja llamada Caja de Transposición (T-box) con 9 bits de entrada y 8 bits de salida. La caja de sustitución S_0 opera los cuatro bits más significativos (izquierda), y S_1 opera sobre los menos significativos (derecha), de cada byte. La salida de las cajas de sustitución es concatenada y al resultado se le aplica la función XOR junto a las sub-llaves, en una operación llamada interrupción de llave (key interruption). Como último paso, la función F permuta los bits de salida, para ello realiza dos pasos: primero, cada byte sufre una permutación, P, luego, los bits son mezclados entre los bytes– cada bit ocupa un byte diferente en la misma posición en que estaba en el byte original, a esto se le llama difusión. Se denota P como el producto de las dos permutaciones antes descritas.

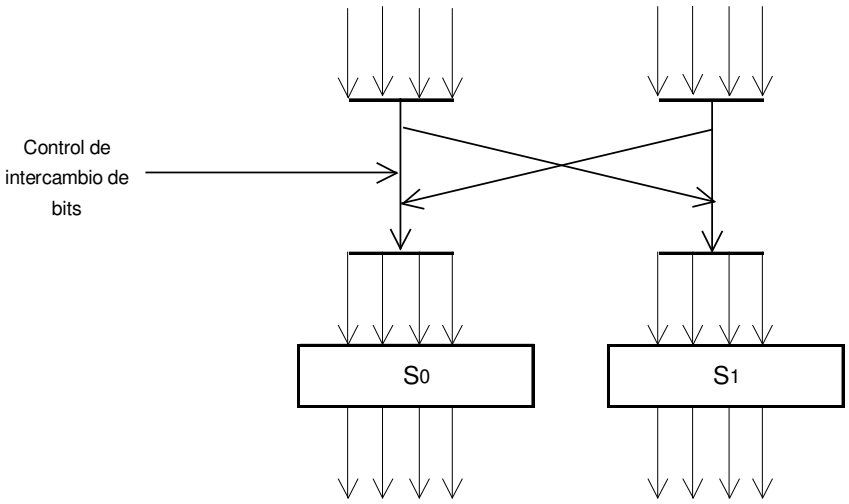


Figura 4.5 Caja de Transposición de Lucifer

A diferencia del DES, no hay intercambio entre vueltas y tampoco son utilizadas las mitades de los bloques.

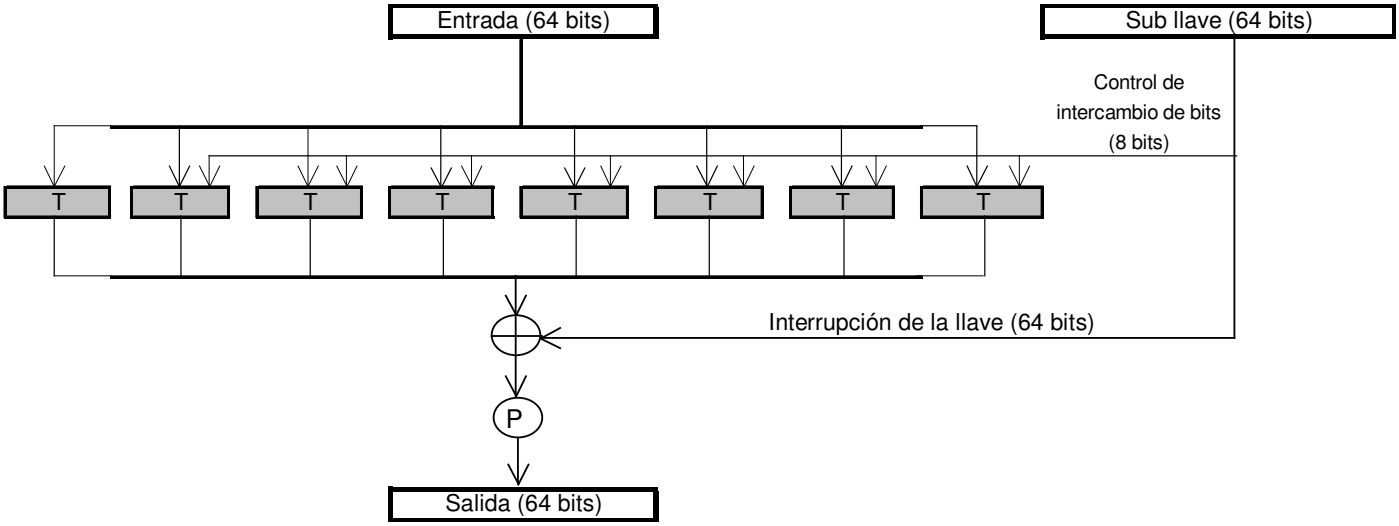


Figura 4.6 La función F de Lucifer

4.2.3 Ataques a Lucifer

Utilizando criptoanálisis diferencial contra la primera versión de Lucifer, los investigadores Eli Biham y Adi Shamir, demostraron que Lucifer puede ser roto[51]. Las condiciones para que tal ruptura sea efectiva son las siguientes:

- Bloques de 32 bits y 8 vueltas, puede ser roto con 40 textos claros elegidos, y 2^{29} pasos; [52]
- Bloques de 128 bits y 8 vueltas con 60 textos claros elegidos y 2^{53} pasos.

Todos estos ataques utilizan las cajas de sustitución del DES.

4.3 DATA ENCRYPTION STANDARD (DES)

El Data Ecryption Standar (DES) ha sido un estándar mundial por cerca de 30 años. A principios de los años 70 dio inicio la búsqueda de un sistema de cifrado no militar. Pocos documentos de investigación eran publicados al respecto. La mayoría de personas sabían que los militares utilizaban equipos para codificación sus comunicaciones, pero pocos entendían la ciencia de la criptografía. La National Security Agency (NSA) tenia información considerable, pero ni siquiera públicamente admitían su propia existencia.

---

[51] Biham, E., Shamir, A. *Differential Cryptanalysis of the Data Encryption Standard*. Proceedings, Berlin. Springer Verlag. 1993.
Biham, E. *New Types of Cryptanalytic Attacks Using Related Keys*. In Tor Helleseth, editor, Advances in Cryptoglogy –EUROCRYPT´93, volume 765 of Lecture Notes in Computer Science, pages 398-409. Springer-Verlag, 1993.
Biham, E., Shamir, A. *Differential Cryptanalysis of DES-like Cryptosystems* (Extended Abstract). The Weizmann Institute of Science. Department of Applied Mathematics. S/f.

[52] Ver Anexo I

En 1972, el National Bureau of Standars (NST) que luego se convirtió en National Institute of Standars and Technology (NIST), inició un programa para proteger las computadoras y la información en las comunicaciones. Como parte del programa, querían desarrollar un algoritmo criptográfico estándar. Un solo algoritmo que pudiera ser probado, certificado, y que además los diferentes equipos que lo utilizaran pudiesen ínter operar.

El 15 de mayo de 1973, el NST solicitó propuestas para el nuevo estándar de algoritmo criptográfico, especificando una serie de criterios que se enumeran a continuación. El algoritmo debía:

- Proveer un alto nivel de seguridad
- Ser completamente especificado y fácil de entender
- Contar con una seguridad que resida en la llave y que no dependa de mantener el algoritmo en secreto
- Estar disponible para todos los usuarios
- Ser adaptable para ser usado en diferentes aplicaciones
- Ser económicamente implementable en dispositivos electrónicos
- Ser eficiente
- Estar dispuesto a ser validado
- Ser exportable

La respuesta del público indicó que había un interés considerable en establecer un estándar criptográfico, pero muy pocos especialistas en el campo. Ninguna de las propuestas recibidas cumplieron los requerimientos.

En agosto de 1974, la National Bureau of Standard (NBS) hizo una segunda solicitud, y esta vez si recibieron un candidato promisorio: un algoritmo *Lucifer*, cuya descripción se abordó en una parte anterior de este documento.

La NBS solicitó ayuda a la NSA para evaluar la seguridad del algoritmo y determinar su vulnerabilidad y ajuste como estándar federal. IBM ya había patentado su idea, pero estuvo dispuesta a poner a disposición de otros su propiedad intelectual. Eventualmente, la NBS trabajó en los términos para un acuerdo con IBM, de manera que recibió una licencia para la creación, uso y venta de equipo para implementar el algoritmo.

Finalmente, el 17 de marzo de 1975, la NBS publicó los detalles del algoritmo y el acuerdo logrado con IBM en el cual ésta última cedía su exclusividad del algoritmo, además solicitaron comentarios del público.

Algunos de los comentarios recibidos se referían a la preocupación de que la NSA hubiese modificado el algoritmo para generar una trampa. Hubo quejas en cuanto a que la NSA había reducido el tamaño de la llave de 128 a 56 bits. Muchos de estos argumentos resultaban angustiosos, pero se aclararon a inicios de los años 90.

En 1976, NBS sostuvo dos talleres para evaluar el estándar propuesto. El primer taller discutió la matemática del algoritmo y la posibilidad de una trampa. En el segundo taller se discutió la posibilidad de incrementar la longitud de la llave. Los diseñadores del algoritmo, evaluadores, implementadores, vendedores, usuarios y críticos fueron invitados.

A pesar de las críticas, el DES fue adoptado como un estándar federal el 23 de noviembre de 1976 y autorizado para ser utilizado en todas las comunicaciones gubernamentales no clasificadas. La descripción oficial del estándar, FIPS[53] PUB 46 "*Data Encryption Standard*" fue publicada el 15 de enero de 1977 y se hizo efectiva seis meses más tarde. El documento FIPS PUB 81 *"DES Modes of Operation*" fue publicado en 1980 y en 1981 se publicó FIPS PUB 74, "*Guidelines for Implementing and Using the NBS Data Encryption Stardard"*.

Según Bruce Schneier, la NSA creyó que el DES era un algoritmo sólo para hadware, pero la NBS publicó suficientes detalles, de tal forma que las personas podían producir software que lo utilizara. De manera no oficial la NSA clasificó al DES como uno de sus errores más grandes. De igual forma, Schneier sostiene que si la NSA hubiera sabido que los detalles iban a permitir a las personas el diseño de un software, nunca habrían estado de acuerdo en revelarlos. Ahora había un algoritmo que estudiar: uno que la NSA había dicho era seguro. No por casualidad el siguiente algoritmo estándar gubernamental era clasificado[54], llamado *Skipjack*, el cual se describe más adelante.

4.3.1 Establecimiento del estándar

La American National Standards Institute (ANSI) aprobó el DES como un estándar privado en 1981. Lo llamaron Data Encryption Algorithm (DEA). ANSI publicó un estándar para los modos de operación de DEA, un documento similar al presentado por NBS, y un estándar para cifrado en redes que utilizaban DES.

4.3.2 Descripción del DES

El DES es un algoritmo que cifra bloques de 64 bits. Un bloque de 64 bits de texto claro ingresa al algoritmo y un bloque de texto cifrado, también de 64 bits, sale del algoritmo.

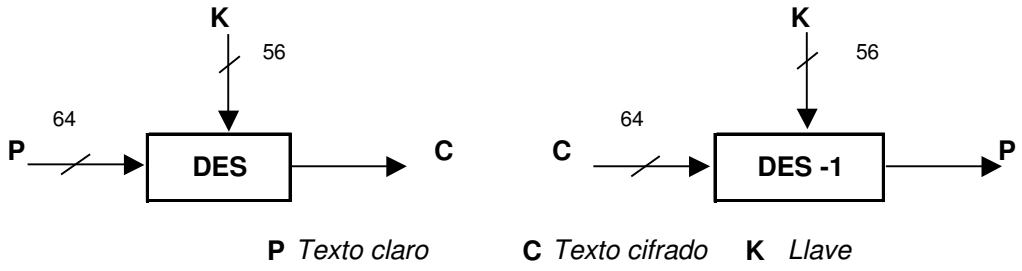


Figura 4.7 Entrada salida del DES

El DES es un algoritmo simétrico, pues el mismo algoritmo y la misma llave son usados para cifrar y descifrar.

Se utilizan llaves de 64 bits, de los cuales 56 contienen la información de la misma, y el resto es utilizado como redundancia para verificación de paridad. Estos bits de paridad son los menos significativos de los bytes que forman las llaves. La llave puede ser cualquier conjunto de 56 bits

[53] FIPS son las siglas de Federal Information Processing Standards Publication, que son los documentos emitidos por el Departamento de comercio de los Estados Unidos a través del NIST.
[54] Schneier, B. Applied Cryptography. Pp. 267. Más adelante se brindan detalles de este estándar.

y puede ser cambiada en cualquier momento. Toda la seguridad del algoritmo descansa en la llave.

En su nivel más simple, el algoritmo no es nada más que una combinación de las dos técnicas básicas de cifrado: confusión y difusión. La construcción fundamental de los bloques del DES es una combinación de estas técnicas sobre el texto, basadas en la llave. Esto es llamado *vuelta*. El DES se compone de 16 vueltas, y aplica la misma combinación de técnicas 16 veces sobre los bloques de texto claro.

El algoritmo utiliza sólo operaciones lógicas y aritméticas sobre grupos de 64 bits, esto contribuyó a que fuera fácilmente implementado en la tecnología de hardware de finales de la década de los setentas. La naturaleza repetitiva del algoritmo lo hace ideal para el uso en chips de propósitos especiales.

4.3.3 Estructura interna del algoritmo

El DES opera sobre bloques de texto claro de 64 bits, después de una permutación inicial, el bloque es partido en sus mitades derecha e izquierda, cada una de 32 bits de longitud. Entonces, los bloques son combinados con las llaves en 16 vueltas con operadores idénticos, llamados *función F*. Después de la decimosexta vuelta, las nuevas mitades derecha e izquierda son concatenadas, y una permutación final (la inversa de la permutación inicial) finaliza el algoritmo.

En cada vuelta, los bits de las llaves son cambiados, luego se seleccionan 48 de los 56 bits que componen la llave. La mitad derecha de los datos es expandida a 48 bits mediante una permutación, combinada con 48 bits de una llave ya permutada y cambiada mediante una función XOR, enviada a través de las 8 cajas de sustitución, produciendo 32 nuevos bits, que serán permutados. Estas cuatro operaciones componen la función *F*. La salida de la función *F* es entonces combinada con la mitad izquierda usando otra función XOR. El resultado de estas operaciones se convierte en la nueva mitad derecha; la mitad derecha anterior se convierte entonces en la nueva mitad izquierda. Estas operaciones son repetidas 16 veces, convirtiéndose así en las 16 vueltas del DES.

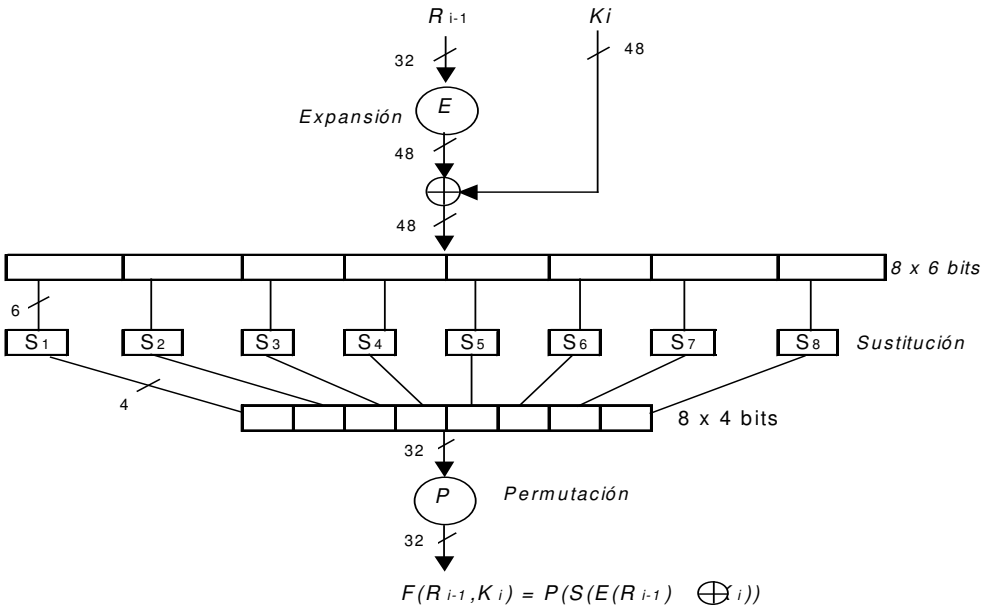


Figura 4.8 Función F

Si B_i es el resultado de la iteración i, L_i y R_i son la mitad izquierda y la mitad derecha de B_i, K_i es la llave de 48 bits de la vuelta i, y *F* es la función que hace las operaciones de sustitución, permutación y XOR con la llave; la vuelta luce como la siguiente figura:

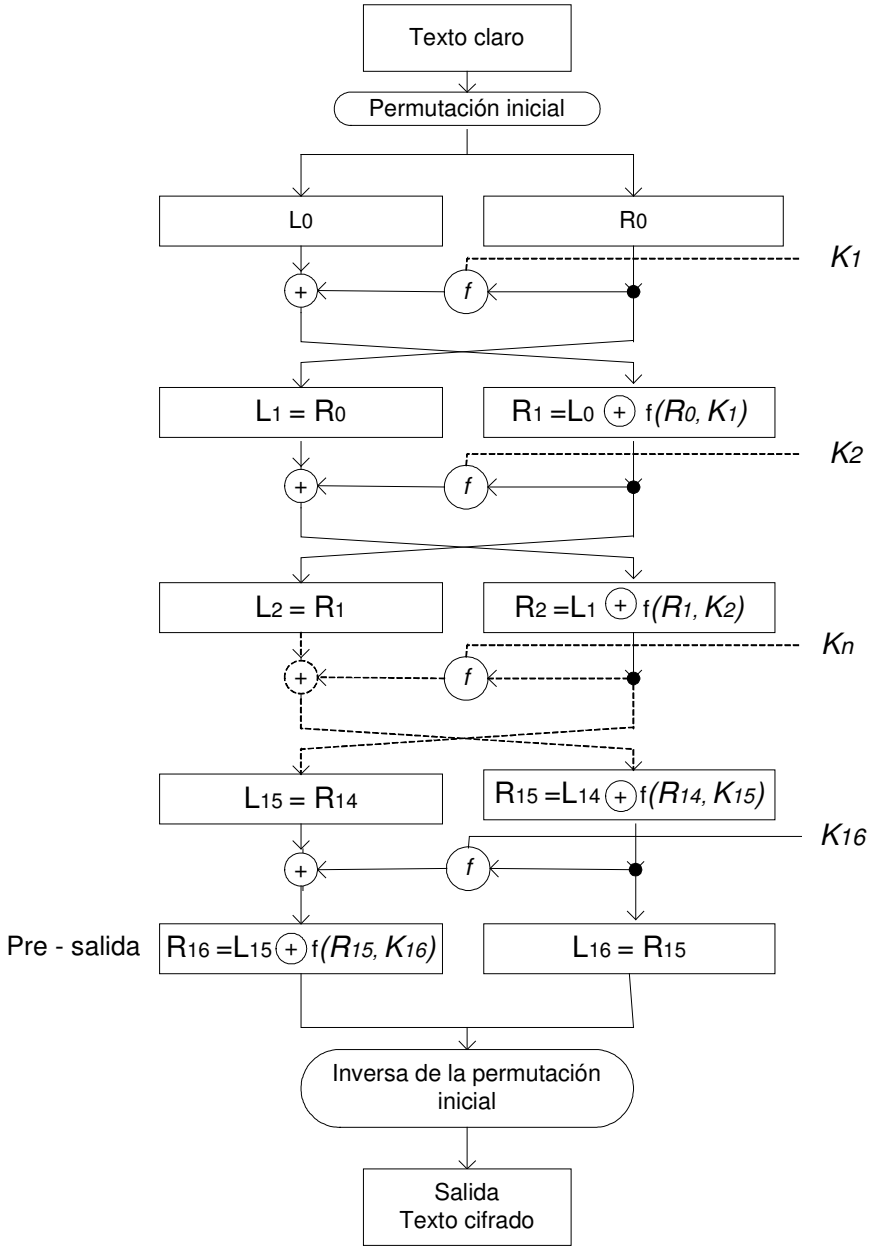


Figura 4.9 Proceso de cifrado utilizando el DES

4.3.4 La permutación inicial

La permutación inicial ocurre antes de la vuelta 1; esta permutación transpone el bloque de entrada, tal como se describe en la tabla 4.7, esta tabla, debe ser leída de izquierda a derecha y de arriba hacia abajo. Por ejemplo, la permutación inicial mueve el bit 58 del texto claro, a la posición del bit 1, el bit 50 a la posición 2, el bit 42 a la posición 3 y así sucesivamente.

La permutación inicial y la correspondiente permutación final no afectan la seguridad del DES.

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

P.I.

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Figura 4.10 Permutación inicial

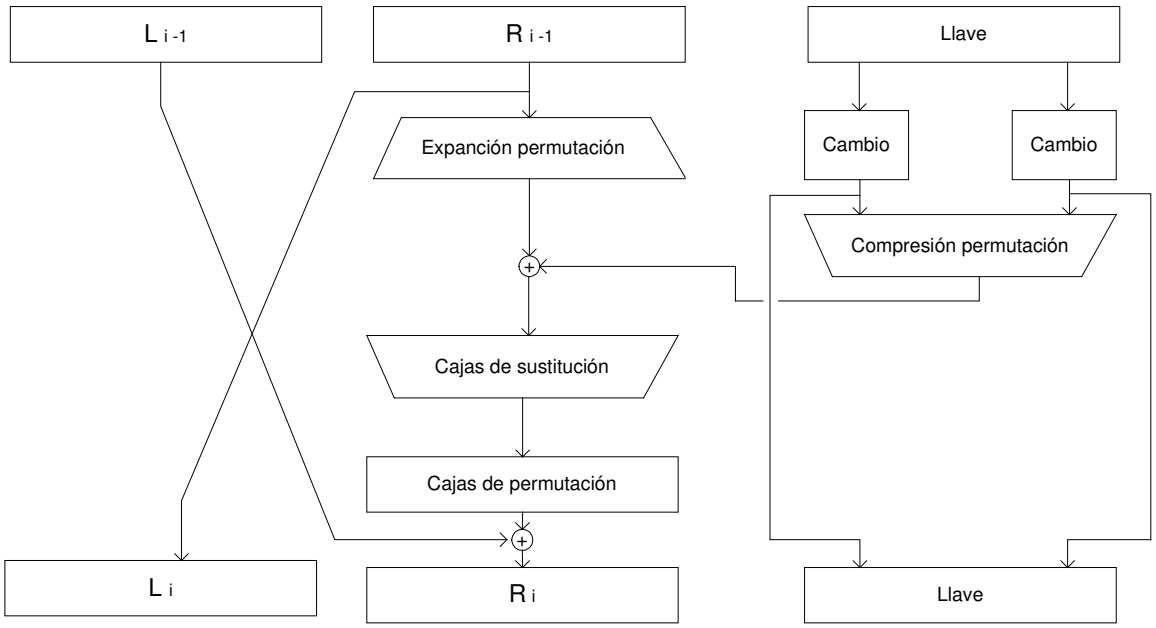


Figura 4.11 Estructura de una vuelta del DES

4.3.5 La transformación de la llave

Inicialmente, los 64 bits de la llave del DES se reducen a una llave de 56 bits al ignorar cada octavo bit. Esto se ilustra en la figura 4.1 Estos bits se utilizan para verificación de paridad, a manera de asegurarse que la llave no tiene errores.

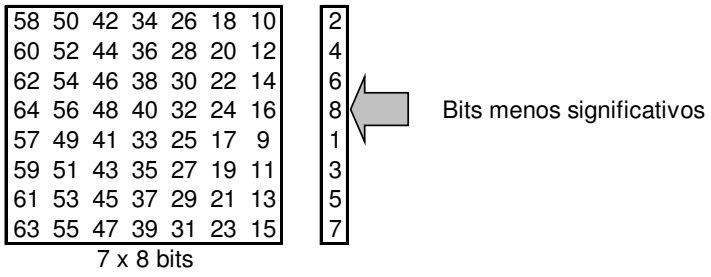


Figura 4.12 Reducción de la llave

Después que la llave de 56 bits es extraída, se genera una sub-llave de 48 bits para cada una de las 16 vueltas del DES. Estas sub-llaves, K_i, son determinadas de la siguiente forma:

- Primero, la llave de 56 bits es dividida en mitades de 28 bits cada una:

58	50	42	34	26	18	10
60	52	44	36	28	20	12
62	54	46	38	30	22	14
64	56	48	40	32	24	16

57	49	41	33	25	17	9
59	51	43	35	27	19	11
61	53	45	37	29	21	13
63	55	47	39	31	23	15

Tabla 4.1 División de la llave

- Posteriormente, las mitades son desplazadas circularmente en sentido izquierdo, uno o dos bits, dependiendo de la vuelta. Estos desplazamientos son mostrados en la tabla siguiente:

Vuelta	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bits desplazados	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Tabla 4.2 Número de vuelta y número de desplazamientos a la izquierda

- Después de haber sido desplazados, 48 de 56 bits son seleccionados. Debido a que esta operación permuta el orden de los bits así como también selecciona un subgrupo de bits, es llamada permutación de compresión. Esta operación provee un subgrupo de 48 bits.

- La figura 4.4 define la permutación de compresión (también llamada *elección permutada*). Por ejemplo, el bit en la posición 33 de la llave cambiada se mueve de la posición 35 de la salida, y el bit de la posición 18 de la llave cambiada es ignorado.

14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

6 x 8 bits

Tabla 4.3 Compresión permutada de la llave

- Debido a estos cambios, un sub-grupo diferente de llaves de bits es usado en cada sub-llave. Cada bit es usado en aproximadamente 14 de las 16 sub-llaves, sin embargo no todos los bits son usados exactamente el mismo número de veces.

4.3.6 La permutación de expansión

Se da por iniciado el procesamiento del bloque de datos de 64 bits. Si el bloque contiene menos de 64 bits deberá ser completado para poder continuar. Seguidamente se realiza la permutación inicial (PI) del bloque:

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

P.I.

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Figura 4.13 Permutación inicial de la llave

Se divide el bloque resultante en mitades de 32 bits cada una. L representará la mitad izquierda y R la mitad derecha, L(0) y R(0).

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8

8 x 4 bits

L(0)

57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

8 x 4 bits

R(0)

Figura 4.14 Mitad izquierda y mitad derecha, L(0) y R(0)

Se denomina *E* a la función que toma un bloque de 32 bits como entrada y genera un bloque de salida de 48 bits, 8 bloques de 6 bits cada uno. Además *i* representa el número de la vuelta.

Se aplican las 16 sub-llaves obtenidas anteriormente, expandiendo, E(R(i)), la mitad de la derecha de los datos, R_i, de 32 a 48 bits, de acuerdo a la siguiente tabla:

32	1	2	3	4	5	4	5
6	7	8	9	8	9	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	1

Tabla 4.4 Posiciones de los bits de salida

Debido a que la operación cambia el orden de los bits así como también repite algunos, esto es conocido como expansión permutación, la cual es a veces llamada la caja E. Para cada bloque de entrada de 4 bits, los primeros y los cuartos bits representan dos bits del bloque de salida, mientas que el segundo y el tercer bit representan un bit del bloque de salida.

Esta operación tiene dos propósitos: hacer la mitad derecha del mismo tamaño de la llave para la operación XOR, y proveer un resultado más largo que puede ser comprimido durante la operación de sustitución. Sin embargo, ninguna de las dos tiene propósitos criptográficos. Al permitir que un bit afecte dos sustituciones, la dependencia de los bits de salida sobre los de entrada, acelera el proceso. Esto es llamado *efecto avalancha*. El DES está diseñado para alcanzar la condición de tener tan rápido como sea posible cada bit correspondiente del texto cifrado, texto claro y la llave.

A pesar que el bloque de salida es más grande que el bloque de entrada, cada bloque de entrada genera un único bloque de salida.

Seguidamente se aplica la función F:

1. XOR al resultado anterior con la llave K(i). E(R(i)) XOR K(i)

2. B(1), B(2),...,B(8). Partiendo de E(R(i-1)) XOR K(i) en ocho bloques de seis bits. B(1) representa a los bits 1 – 6, B(2) representa a los bits 7 – 12,..., B(8) representa los bits 43 – 48.

4.3.7 Las cajas de sustitución

Después que a la llave comprimida se le aplica la función XOR con un bloque expandido, los resultados de 48 bits se mueven a una operación de sustitución. Las sustituciones son llevadas a cabo por ocho cajas de sustitución. Cada caja de sustitución tiene una entrada de 6 bits y una salida de 4.

El total de memoria requerida para las ocho cajas de sustitución del DES es de 256 bytes. Los 48 bits son divididos en ocho sub bloques de 6 bits cada uno. Cada bloque por separado es operado por una caja de sustitución separada: El primer bloque es operado por la caja de sustitución 1, el segundo por la segunda caja, y así sucesivamente.

Cada caja de sustitución es una tabla de 4 filas y 16 columnas. Cada entrada de la caja es un número de 4 bits. La entrada de 6 bits de la caja de sustitución especifica bajo cual número de fila o columna se buscará la salida.

Las cajas de sustitución son la parte crítica del DES. Las otras operaciones del algoritmo son lineales y fáciles de analizar. Las cajas de sustitución son no lineales y más que nada, aportan la seguridad al DES.

Si S_1 es la función definida en la tabla siguiente y B es un bloque de 6 bits, entonces $S_1(B)$ estaría conformado de la siguiente forma:

		Número de fila																*No de caja*
Binario	***Decimal***	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	00	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7	**S1**
1	01	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8	
2	10	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0	
3	11	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13	
0	00	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10	**S2**
1	01	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5	
2	10	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15	
3	11	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9	
0	00	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8	**S3**
1	01	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1	
2	10	13	6	4	9	8	15	3	0	11	1	2	12	5	20	14	7	
3	11	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12	
0	00	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15	**S4**
1	01	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9	
2	10	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4	
3	11	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14	
0	00	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9	**S5**
1	01	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6	
2	10	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14	
3	11	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3	
0	00	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11	**S6**
1	01	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8	
2	10	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6	
3	11	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13	
0	00	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1	**S7**
1	01	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6	
2	10	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2	
3	11	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12	
0	00	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7	**S8**
1	01	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2	
2	10	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8	
3	11	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11	

Tabla 4.5 Cajas de sustitución del DES

El resultado de esta sustitución son ocho bloques de 4 bits, los cuales son recombinados en un solo bloque de 32 bits.

16 7 20 21 29 12 28 17	1 15 23 26 5 18 31 10
2 8 24 14 32 27 3 9	19 13 30 6 22 11 4 25

El bloque anterior pasa a la siguiente etapa: cajas de permutación.

Una vuelta del DES, con la función F en detalle, se ilustra en la siguiente figura:

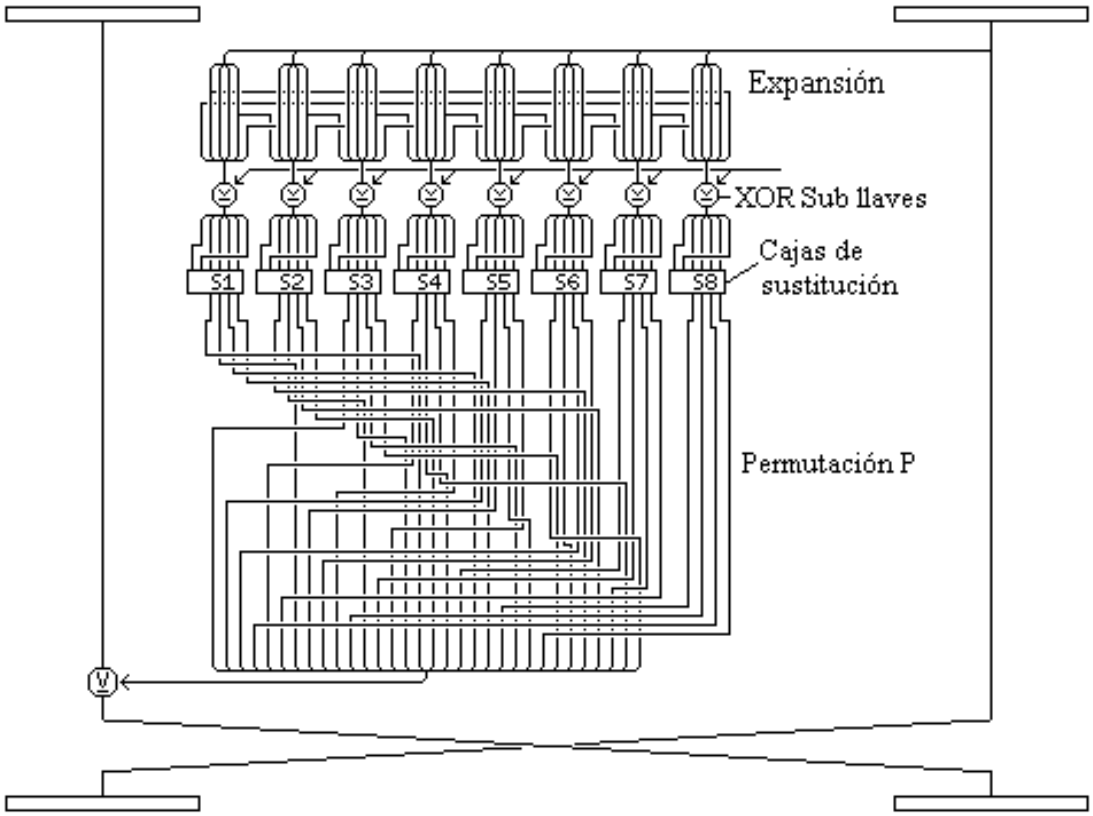


Figura 4.15 Resumen del DES

4.3.8 Las cajas de permutación

La salida de 32 bits de las cajas de sustitución es permutada de acuerdo a las cajas de permutación (P-box). Esta permutación apunta a cada entrada de bit a una posición de salida, ningún bit es usado dos veces y ninguno es ignorado. Esto es llamado permutación directa (straight permutation) o simplemente permutación.

Finalmente, al resultado de una caja de permutación se le aplica la función XOR junto con la mitad izquierda del bloque inicial de 64 bits. Entonces las mitades izquierdas y derechas son intercambiadas y otra vuelta inicia.

4.3.9 La permutación final

La permutación final es la inversa de la permutación inicial y esta descrita en la tabla 4.6

40 8 48 16 56 24 64 32	39 7 47 15 55 23 63 31
38 6 46 14 54 22 62 30	37 5 45 13 53 21 61 29
36 4 44 12 52 20 60 28	35 3 43 11 51 19 59 27
34 2 42 10 50 18 58 26	33 1 41 9 49 17 57 25

Tabla 4.6 Posición de los bits en la permutación inicial

Nótese que las mitades izquierda y derecha no son intercambiadas después de la última vuelta del DES; en su lugar, los bloques concatenados R16L16 son usados como entrada de la permutación final. En esta parte no se da ningún cambio significativo, al cambiar las mitades y realizar los cambios en la permutación brindará el mismo resultado.

4.3.10 Proceso de descifrado del DES

Después de todas las sustituciones, permutaciones, funciones XOR e intercambios, sería fácil pensar que el algoritmo para descifrar sería totalmente diferente al algoritmo de cifrado. Por el contrario, las múltiples operaciones fueron elegidas para producir una propiedad muy útil: el mismo algoritmo trabaja para cifrar y descifrar.

Con el DES es posible utilizar la misma función para cifrar o descifrar un bloque. La única diferencia reside en que las llaves deben ser usadas en el orden inverso. Esto es, si las llaves de cifrado para cada llave son K_1, K_2, K_3..., K_{16}, entonces las llaves para descifrar serán K_{16}, K_{15}, K_{14},..., K_1. El algoritmo que genera la llave usada para cada vuelta es circular también.

4.3.11 Modos de operación del DES

El DES tiene cuatro modos de operación: ECB, CBC, OFB y CFB. Los estándares bancarios, según ANSI, especifican ECB y CBC para cifrado y CBC y CFB para autenticación.

4.3.11.1 ECB (Electronic Code Book)

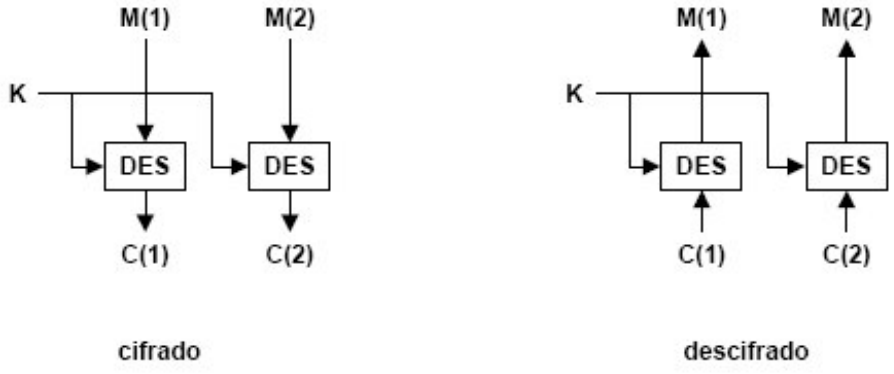


Figura 4.16 Modo de cifrado ECB en el DES
M: Mensaje, C: Texto cifrado, K: Llave

Los datos son divididos en bloques de 64 bits y cada bloque es cifrado, uno a la vez. Cifrados diferentes sobre bloques diferentes los hace independientes. Esto significa que si la información es transmitida a través de una red o una línea telefónica, los errores de transmisión sólo afectarán al bloque que contiene el error, aunque esto también significa que todos los bloques serán ordenados y el error no sería detectado. El ECB es el más débil de todos los modos de cifrado porque no implementa medidas adicionales de seguridad, aparte de las básicas del algoritmo DES. Sin embargo, el ECB es el más rápido y fácil de implementar, haciéndolo el modo de cifrado que más comúnmente se utiliza en aplicaciones comerciales.

4.3.11.2 CBC (Cipher Block Chaining)

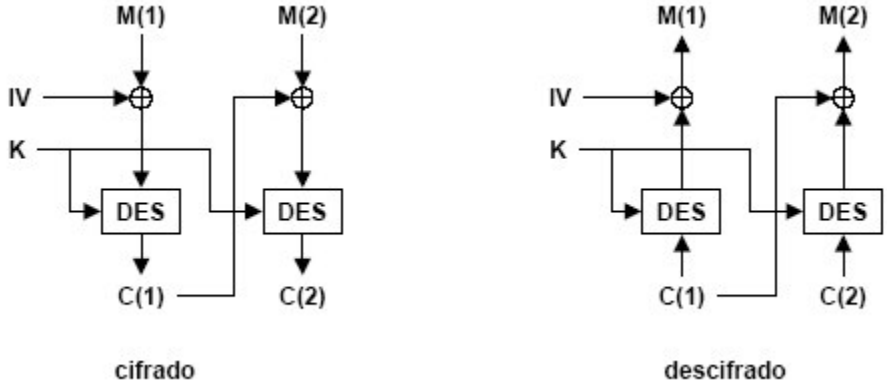


Figura 4.17 Modo de cifrado CBC en el DES
M: Mensaje, IV: Vector de inicialización, C: Texto cifrado, K: Llave

En este modo de operación, a cada bloque cifrado con el modo ECB se le aplica la función XOR con el siguiente bloque de texto claro, para luego ser cifrado, además de hacer que todos los bloques dependan de todos los bloques anteriores. Esto significa que para encontrar el texto claro de un bloque en particular, se necesitaría conocer el texto cifrado, la llave, y el texto cifrado del bloque anterior. El primer bloque a ser cifrado no tendrá texto cifrado previo, de manera que al texto claro se le aplica una función XOR junto a un número de 64 bits llamado Vector de Inicialización, o IV. Así, sí los datos son transmitidos a través de la red o línea telefónica y ocurre un error de transmisión, el error será cargado por todos los bloques siguientes, ya que cada bloque es dependiente del anterior. Este modo de operación es más seguro que el ECB porque tiene un paso extra, la operación XOR que agrega más capas al proceso de cifrado.

4.3.11.3 CFB (Cipher Feedback)

En este modo de operación, los bloques de texto claro cuya longitud sea menor a 64 bits pueden ser cifrados. Normalmente, tiene que realizarse un proceso especial para manipular los archivos cuyo tamaño no es un perfecto múltiplo de 8 bytes, pero este modo elimina dicha necesidad. Un bloque de 64 bits llamado registro de desplazamiento, es usado como entrada de texto claro para el DES, este es un valor arbitrario y cifrado con el algoritmo. Seguidamente, el texto cifrado es hecho pasar a través de un componente extra llamado *M-box*, el cual simplemente selecciona los *M* bits más significativos del texto cifrado, donde *M* es el número de bits en el bloque que se desea cifrar, al valor obtenido, se le aplica la función XOR junto con el texto claro real, y la salida producida es el texto cifrado. Finalmente, el texto cifrado es ingresado en el registro de desplazamiento, y utilizado como la semilla de texto claro para los siguientes bloques que serán cifrados.

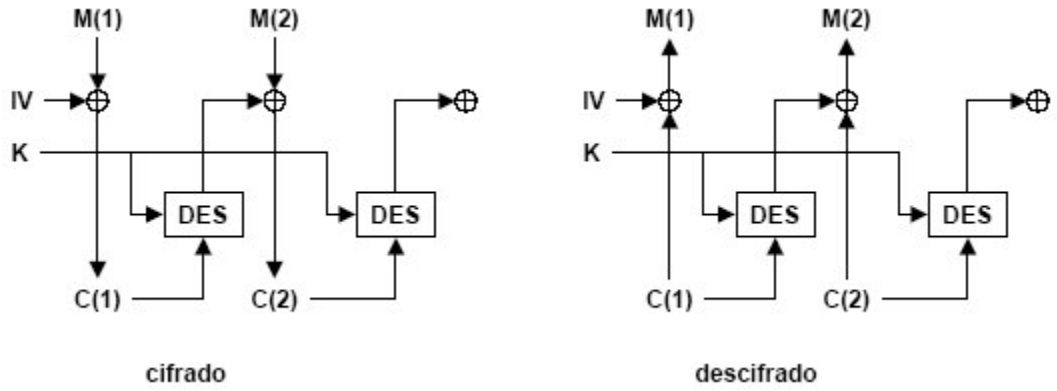


Figura 4.18 Modo de cifrado CFB en el DES
M: Mensaje, IV: Vector de inicialización, C: Texto cifrado, K: Llave

Tal como en el modo CBC, un error en un bloque afectaría a todos los bloques subsecuentes durante la transmisión de datos. Este modo de operación es similar al CBC y es muy seguro, aunque más lento que el ECB debido a la complejidad agregada.

4.3.11.4 OFB (Output Feedback)

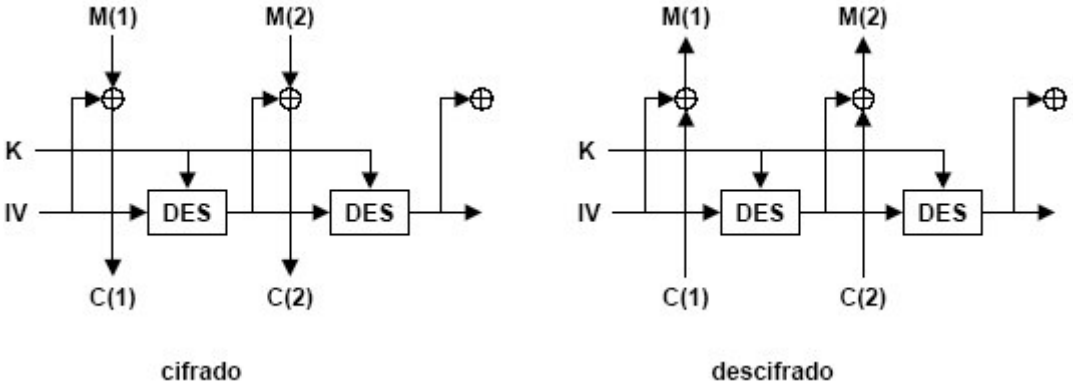


Figura 4.19 Modo de cifrado OFB en el DES
M: Mensaje, IV: Vector de inicialización, C: Texto cifrado, K: Llave

Este modo de operación es similar al modo CFB, con la diferencia que el texto cifrado de salida del DES es nuevamente ingresado en el registro de desplazamiento, en lugar del texto cifrado actual. El registro de desplazamiento inicializado con un valor inicial arbitrario, para luego pasar por el algoritmo DES. La salida del DES se hace pasar a través de una M-box para luego alimentar el registro de desplazamiento, a manera de prepararse para el siguiente bloque. Al resultado se le aplica la función XOR junto al texto claro real (el cual debe ser menor de 64 bits tal como en el modo CFB), el resultado obtenido será el texto cifrado. A diferencia de los modos CFB y CBC, un error en la transmisión de un bloque no afectará los subsecuentes, porque una vez se tenga el valor inicial del registro de desplazamiento, éste continuará generando nuevas entradas de texto claro al registro de desplazamiento sin ningún otro dato de entrada. Sin embargo, este modo de operación es menos seguro que el CFB porque sólo es necesitado el texto real y el texto cifrado por el DES para encontrar el texto claro del bloque más reciente.

4.3.12 Seguridad del DES

Ha habido mucha especulación acerca de la seguridad del DES, en cuanto a la longitud de la llave, número de iteraciones y el diseño de las cajas de sustitución. Las cajas de sustitución son particularmente misteriosas; todas ellas constantes, sin ninguna razón aparente del porqué o para qué son así. A pesar de que IBM sostenía que el funcionamiento interno del DES era el resultado de 17 años-hombre de criptoanálisis intensivo, algunas personas temían que la NSA hubiera puesto una trampa en el algoritmo de manera que ellos pudieran fácilmente descifrar los mensajes.

El Comité de Inteligencia de los Estados Unidos, investigó el asunto en 1978. Los resultados de la investigación fueron clasificados, pero un resumen desclasificado de los hallazgos exoneraba a la NSA de cualquier manipulación del diseño del algoritmo. *"Se dijo que se convenció a IBM que utilizar una llave más corta era lo más adecuado, además de haber asistido indirectamente la estructura y el desarrollo de las cajas de sustitución"*. Sin embargo, como el gobierno no hizo público el resultado de la investigación, muchas personas permanecieron con las dudas.

La NSA, al ser consultada sobre alguna debilidad impuesta al DES, dijo *" Con respecto al Data Encryption Standard (DES), creemos que el informe público del Senado en 1978, acerca del rol de la NSA en el desarrollo del DES responde a su pregunta*".

Schneier reporta el hecho de que dos de los criptógrafos que diseñaron el DES, Tuchman y Meyer, dijeron que la NSA no alteró el diseño. Adicionalmente, Coppersmith escribió: "*La NSA también brindó consejos técnicos a IBM*". Konheim agregó "*Nosotros enviamos las cajas de sustitución a Washington, y éstas regresaron totalmente cambiadas. Hicimos nuestras evaluaciones y las pasaron*". Algunas personas toman este comentario como una evidencia de que la NSA había manipulado el algoritmo.[55]

Entonces la pregunta es ¿porque modificaron las cajas de sustitución?. Tal vez fue para asegurarse que la IBM no había puesto una trampa en el DES. La NSA no tenía ninguna razón para confiar en los investigadores de IBM, y era su responsabilidad asegurarse que el DES estuviera libre de trampas. Modificar las cajas de sustitución era una forma de asegurarse.

4.3.13 Llaves débiles del DES

Debido a la forma en que la llave inicial es modificada para obtener una sub-llave en cada vuelta del algoritmo, ciertas llaves iniciales son consideradas débiles. Se debe recordar que el valor inicial es partido por mitad, y que cada mitad es desplazada de forma independiente. Si todos los bits en cada mitad son ceros o unos, la llave usada para cada ciclo del algoritmo es la misma para el resto de ellas. Esto puede ocurrir si la llave está formada sólo por unos, solo por ceros o sí una mitad de la llave esta compuesta de unos y la otra por ceros. También dos de las llaves débiles tienen otras propiedades que las hacen menos seguras.

Las cuatro llaves débiles del DES son mostradas en notación hexadecimal en la Tabla 4.7, reacuérdese que cada octavo bit es para paridad.

Existen pocas llaves que son consideradas débiles para el DES. El uso de estas llaves puede reducir la efectividad de la seguridad brindada por el TDES y deberá ser evitada.

[55] Schneier, B. Applied Cryptography. Pp. 278 y 280.

```
0000000 0000000
0000000 FFFFFFF
FFFFFFF 0000000
FFFFFFF FFFFFFF
```

Tabla 4.7 Llaves débiles del DES

Adicionalmente, algunos pares de llaves producen textos cifrados iguales. En otras palabras, una llave de un par puede descifrar mensajes cifrados con la otra llave del par. Esto se debe a la forma en que el DES genera las sub-llaves; en lugar de generar 16 sub-llaves diferentes, estas llaves generan sólo dos. Cada una de estas sub-llaves es usada ocho veces en el algoritmo. Estas llaves son consideradas *semi-débiles*, y son mostradas en notación hexadecimal a continuación:

```
 01FE O1FE 01FE O1FE y FE01 FE01 FE01 FE01
1FE0 1FE0 0EF1 0EF1 y E01F E01F F10E F10E
01E0 01E0 01F1 01F1 y E001 E001 F101 F101
1FFE 1FFE 0EFE 0EFE y FE1F FE1F FE0E FE0E
011F 011F 010E 010E y 1F01 1F01 0E01 0E01
E0FE E0FE F1FE F1FE y FEE0 FEE0 FEF1 FEF1
```

Tabla 4.8 Pares de llaves semi-débiles del DES

Antes de condenar DES por tener llaves débiles, considérese que esta lista de 64 llaves es minúscula comparada con la lista total de 72,057,594,037,927,936 llaves posibles. Si se selecciona una llave de manera aleatoria, las posibilidades de elegir una de las débiles es muy poca.

4.3.14 Longitud de la llave

La propuesta original de IBM para NBS tenía una llave de 128 bits. En el tiempo que el DES se hizo un estándar, ésta fue reducida a 56 bits. Muchos criptógrafos discutieron por la longitud de la llave. Sus argumentos se centraron en la posibilidad de ataques de fuerza bruta.

4.3.15 Críticas al diseño del DES

Numero de vueltas

¿Porqué 16 vueltas?, ¿porqué no 32?. Después de cinco vueltas, cada bit del texto cifrado es una función de cada bit del texto claro y de cada bit de la llave, y después de ocho vueltas el texto cifrado es esencialmente una función aleatoria de cada bit del texto claro y cada bit de cada bit. Cambios mínimos en el texto claro o la llave tienen un efecto esperado de cambios de alrededor del 50% sobre los bits que se procesan. Tal como se había explicado previamente, recibe el nombre de efecto avalancha[56]. Entonces, ¿porqué no detener el proceso después de las ocho vueltas? A lo largo de los años, variantes del DES han sido creados, en ellos se ha comprobado que con un número reducido de vueltas los ataques son exitosos. El DES con tres o cuatro vueltas fue fácilmente atacado en 1982. El DES con seis vueltas cayó algunos años más tarde. El criptoanálisis de Biham y Shamir explica que el DES con un número menor a 16 vueltas pudo ser roto con un ataque de texto claro conocido, más eficientemente que utilizando la fuerza bruta[57].

[56] Bauer, F. Op. Cit. pp. 174.
[57] Ver Anexo I.

Diseño de las cajas de sustitución

Además de haber sido acusados de reducir el tamaño de la llave, la NSA fue también acusada de modificar el contenido de las cajas de sustitución. Cuando fue presionada para brindar una justificación para las cajas de sustitución, la NSA indicó que los elementos del diseño del algoritmo fueron "sensibles" y que no se podían hacer públicos. Muchos criptógrafos estaban conscientes de la NSA diseñó un agujero de seguridad en las cajas de sustitución, haciendo posible que ellos pudieran fácilmente criptoanalizar el algoritmo.

Desde entonces, un considerable esfuerzo se ha hecho para analizar el diseño y operación de las cajas de sustitución. A mediados de los años 70, la Corporación Lexar y los Laboratorios Bell examinaron la operación de las cajas de sustitución. Ninguno de los análisis reveló alguna debilidad, pero, ambas instituciones encontraron situaciones inexplicables: las cajas de sustitución tenían más rasgos en común con la *transformación lineal* de lo que ellos esperaban. Los Laboratorios Bell argumentaron que las cajas de sustitución podrían tener trampas escondidas, y el reporte de los laboratorios Lexar concluyó:*"Algunas estructuras han sido encontradas en el DES, las cuales, indudablemente, fueron insertadas para reforzar el sistema en contra de ciertos tipos de ataque. También se han encontrado estructuras que aparentemente lo debilitan".*

Por otro lado, este reporte también advertía:*"El problema (de la búsqueda de la estructura de las cajas de sustitución) es complicado debido a la habilidad de la mente humana de encontrar estructuras aparentes en datos aleatorios, lo cual no es en realidad una estructura".*

En una segunda convención del DES, la NSA reveló varios criterios de diseño de las cajas de sustitución. Esto no aclaró las dudas de las personas, y el debate continúa.

Algunos aspectos curiosos acerca de las cajas de sustitución son resaltadas en diferentes investigaciones:

- Los últimos tres bits de salida de la cuarta caja de sustitución pueden ser derivados de la misma forma que la primera, al complementar algunos de los bits de entrada.

- Dos entradas diferentes de las cajas de sustitución, pueden producir la misma salida. Es posible obtener la misma salida de una vuelta del DES al cambiar los bits en solo tres cajas continuas.

- Shamir notó que las entradas de las cajas de sustitución parecían ser, de alguna manera, desbalanceadas, pero ello no podía ser una base para un ataque. Él mencionó un rasgo de la caja de sustitución número cinco, pero tomó otros ocho años antes que el criptoanálisis la explotara.

4.3.16 Resultados adicionales

Existen otros intentos de analizar el DES. Un criptógrafo utilizó un análisis no aleatorio basado en espectros. Otros analizaron secuencias de factores lineales, pero sus ataques fallaron después

de la octava vuelta. En 1987 un ataque realizado por Donald Davies explotó la forma en que la expansión permutación repite los bits en cajas de sustitución adyacentes, este ataque igual falla después de la octava vuelta.

Criterios cajas de sustitución y permutación

Después que el *criptoanálisis diferencial*[58] se hiciera público, IBM publicó el criterio de diseño para las cajas de sustitución y las cajas de permutación, estos son:

- Cada caja de sustitución tiene 6 bits de entrada y 4 bits de salida. (Este era el tamaño máximo que podía acomodar en un solo chip con la tecnología de 1974).

- Ningún bit de salida de una caja de sustitución debería estar tan cercano a una función lineal de los bits de entrada.

- Si se dejan fijos los bits de la derecha y los de la izquierda, de manera que sólo varíen los bits del medio, cada salida posible de 4 bits será obtenida solamente una vez.

- Sí dos entradas de una caja de sustitución difieren exactamente en un bit, las salidas deberán diferir en por lo menos 2 bits.

- Si dos entradas de una caja de sustitución difieren exactamente en sus dos primeros bits y son idénticas en sus últimos 2 bits, las dos salidas no deberán ser las mismas.

Criterios para las cajas de permutación:

- Los cuatro bits de salida de cada caja de sustitución en la vuelta *i* están distribuidos de manera que dos de ellos afectan los bits de en medio de las cajas de sustitución en la vuelta $i + 1$ y los otros dos afectan los bits finales.

- Los cuatro bits de salida de cada caja de sustitución afectan seis diferentes cajas de sustitución, nunca dos afectan la misma caja.

- Si el bit de salida de una caja de sustitución afecta un bit de en medio de otra, entonces un bit de salida de otra caja de sustitución no puede afectar el bit de en medio de la primera caja de sustitución.

Actualmente, generar cajas de sustitución puede ser muy fácil, pero era una tarea complicada a principio de los años 70. Tuchman dijo que ellos ejecutaban durante meses los programas que generaban las cajas de sustitución.

4.3.17 Variantes del DES

4.3.17.1 Triple DES

Desde la presentación del DES, a mediados de los años 70, los criptoanalistas han estado conscientes de la debilidad de su llave de 56 bits. En 1977 Diffie y Hellman estimaron el costo de una máquina capaz de recuperar una llave de 56 bits en un día, US$20 millones. En 1993 se

[58] Ver Anexo I.

presentó el diseño de una máquina que reduciría la búsqueda a 3.5 horas. El diseño consistía en 57,000 circuitos y tenía un costo estimado de 1 millón de dólares. Media década más tarde, la EFF, Electronic Frontier Foundation, construyó la primera máquina para tal fin. El costo fue de $250,000 y se le llamó "Deep Crack", la cual hizo la búsqueda de las llaves y tardó 9 días.

Una vez quedó claro que el DES no proveía una seguridad adecuada debido a su llave de 56 bits, el algoritmo fue gradualmente reemplazado por el llamado Triple DES, también conocido como TDEA. La idea del funcionamiento del algoritmo fue propuesta por Diffie y Hellman, quienes notaron que al realizar el doble de las operaciones del DES no obtenían ningún incremento en su seguridad. Matyas y Merkle sugirieron una lista de esquemas de reemplazo para el DES el cual fue incluido en el estándar ANSI X9.52, el cual en 1999 reemplazaba al DES. Actualmente el TDES esta siendo reemplazado por el *Advanced Encryption Standard (AES)*. La transición es lenta, sin embargo, se espera que ambos esquemas de cifrado coexistan por muchos años más.

El TDES consiste en aplicar tres veces el DES, en el ANSI X9.52, se define de la siguiente forma:

$$C = EK3\ (DK2\ (EK1\ (P)))$$

Donde P y C son textos claros de 64 bits y textos cifrados, y EK(·) and DK(·) denotan las funciones de cifrado y descifrado del DES. El estándar especifica tres formas diferentes de elegir las llaves de 56 bits, K1, K2, y K3:

1. K1, K2, y K3 son independientes, K1≠K2 ≠K3 ≠K1, (168 bits);
2. K1 ≠ K2 y K3 = K1.
3. K1 = K2 = K3 (56 bits).

La tercera opción es equivalente al DES y provee compatibilidad con sistemas de cifrado anteriores.

La opción 2 duplica el número de bits de la llave y triplica el número de vueltas del DES. La opción 1 triplica ambos. Como resultado, el algoritmo es reforzado ante ataques criptográficos y también ante búsquedas exhaustivas de la llave.

El triple DES tiene la ventaja de utilizar llaves más largas que el DES, pero sigue operando con bloques de 64 bits. Esta limitación, en conjunto con la realidad que el TDES es lento, motiva a cambiarse al AES que utiliza bloques de 128 bits.

Las llaves del TDES deben ser manipuladas de acuerdo a la publicación especial de la NIST, SP 800-72, "*Recommendation for Key Managements"*. Las siguientes especificaciones para el manejo de llaves deben ser cumplidas al momento de la implementación del algoritmo.

Requerimientos para las llaves

Para todos los modos de operación de TDEA, tres llaves K1, K2, K3 definen el esquema, los requerimientos que las llaves deben cumplir son:

- Ser secreta
- Ser generada aleatoria o pseudoaleatoriamente
- Ser independiente

- Tener integridad, es decir, cada llave no debe haber sido alterada de forma no autorizada desde que fue generada, transmitida o almacenada por una fuente autorizada.
- Ser utilizada en el orden apropiado

Ser considerada una cantidad fija de manera que, una llave individual no pueda ser manipulada mientras se dejan las otras dos sin cambios

4.3.17.2 DES con sub-llaves independientes

Otra variación resulta al utilizar diferentes sub-llaves para cada vuelta, en lugar de generarlas desde una sola llave de 56 bits. Los 48 bits son usados en cada una de las 16 vueltas, esto significa que la longitud de la llave sería de 768 bits. Esta variante incrementará drásticamente la dificultad de un *ataque de fuerza bruta* contra el algoritmo; dicho ataque tendría una complejidad de 2^{768}.

Aunque, las sub-llaves independientes contrarrestan el *criptoanálisis lineal*, esta variante es susceptible al *criptoanálisis diferencial* y puede ser roto con 2^{61} textos claros elegidos. Parecería que cualquier modificación al manejo y generación de llaves no puede hacer al DES más fuerte.

4.3.17.3 DESX

El DESX es una variante del DES, que ha sido incluido en el programa *Mail Safe* desde 1986 y en el grupo de herramientas *BSAFE* desde 1987. DESX utiliza una técnica llamada blanqueamiento para confundir las entradas y salidas del DES. Además de una llave de 56 bits, el algoritmo tiene una llave adicional de blanqueamiento de 64 bits. A estos 64 bits se les aplica una función XOR junto al texto claro, antes de la primera vuelta del DES. Junto al texto cifrado se aplica la función XOR junto a 64 bits adicionales. El blanqueamiento hace que el DESX sea más fuerte que el DES ante un ataque de fuerza bruta; el ataque requiere $(2^{120})/n$ operaciones con n textos claros conocidos. Esto también mejora la seguridad ante un criptoanálisis lineal y diferencial; los ataques requieren 2^{61} textos claros elegidos y 2^{60} textos claros conocidos respectivamente.

4.3.17.4 CRYPT(3)

CRYPT(3) es una variante del DES encontrada en sistemas de UNIX. Es principalmente usada como una función para contraseñas, pero algunas veces puede ser usada para cifrado. La diferencia entre CRYPT(3) y DES es que el primero tiene una llave independiente para la expansión permutación con 2^{12} posibles permutaciones.

4.3.17.5 DES generalizado

El DES generalizado o GDES fue diseñado para acelerar el DES y para reforzar el algoritmo.

GDES opera con bloques de una variable de texto claro. Bloques cifrados son divididos hasta en sub bloques de 32 bits; el número exacto depende del tamaño del bloque total.

La función *F* es calculada una vez en cada ronda con cada bloque que se encuentra más a la derecha. Al resultado se le aplica una función XOR con todas las otras partes, las cuales son rotadas hacia la derecha. Este algoritmo tiene un número variable de vueltas, *n*. Hay una pequeña modificación en la última vuelta, de manera que el proceso de cifrado y descifrado difieren únicamente por el orden de las sub-llaves (igual que el DES).

4.3.17.6 DES con cajas de sustitución alternativas

Otras modificaciones realizadas al DES se centran en las cajas de sustitución. Algunos diseños hacen que el orden de las cajas sea variable. Otros diseñadores varían por sí mismos el contenido de las cajas de sustitución. Biham y Shamir demostraron que el diseño de las cajas de sustitución, y aún el orden de las cajas, fue optimizado contra el criptoanálisis diferencial:*"El reemplazo del orden de las ocho cajas del DES (sin cambiar su valor) hace que el DES sea mucho más fácil: El DES con 16 vueltas de un orden reemplazado en particular es rompible en alrededor de* 2^{38} *pasos. DES con cajas de sustitución alternas ha demostrado ser muy fácil de romper. Aún con cambios mínimos de una entrada en una de las cajas de sustitución puede hacerlo fácil de romper".*

Las cajas de sustitución del DES no fueron optimizadas contra el criptoanálisis lineal.

4.3.17.7 RDES

RDES es una variante que reemplaza la mitad izquierda y derecha al final de cada vuelta con un intercambio que depende de una llave. Este intercambio es fijo, dependiendo solamente de la llave. Esto quiere decir que los 15 intercambios dependientes de las llaves ocurren con 2^{15} posibles instancias, y que esa variante no es resistente a un criptoanálisis diferencial.

Este algoritmo tiene una gran cantidad de número de llaves débiles. En realidad, casi todas las llaves son más débiles que una llave típica del DES. Esta variante no debe ser utilizad

4.3.17.8 S^nDES

Un grupo de investigadores coreanos, dirigidos por Kwangjo Kim, intentaron encontrar un grupo de cajas de sustitución que fueran óptimamente seguras en contra de criptoanálisis diferenciales y lineales. Su primer intento, conocido como s^2DES, fue presentado y demostró ser peor que el DES en contra de criptoanálisis diferencial. Su siguiente intento, s^3DES, fue representado y demostró ser peor en contra del criptoanálisis lineal. Biham sugirió un cambio menor en el s^3DES para hacerlo resistente en ambos tipos de criptoanálisis. El grupo desarrolló técnicas nuevas para el diseño de cajas de sustitución. Ellos propusieron s^nDES.

4.3.17.9 DES con cajas dependientes de llaves

El criptoanálisis lineal y diferencial funciona solamente sí el analista conoce la composición de las cajas de sustitución. Si las cajas de sustitución son dependientes de las llaves y elegidas por un método criptográfico más fuerte, entonces el criptoanálisis lineal y diferencial son mucho más difíciles.

4.3.18 ¿Qué tan seguro es DES ahora?

La respuesta es fácil y difícil. La parte fácil es que basta con tan solo mirar la longitud de la llave. Una máquina que realice un ataque de fuerza bruta al DES y que pueda encontrar una llave en un promedio de 3.5 horas costaba en 1993 alrededor de un millón de dólares.

El DES es tan popular que sería inocente creer que NSA y sus contrapartes no la hayan construido. También hay que recordar que el costo de dicha máquina caerá en un factor de 5 cada 10 años. En otras palabras el DES se vuelve más débil según el tiempo va pasando.
La parte difícil de la respuesta trata de estimar las técnicas de criptoanálisis. El criptoanálisis diferencial fue conocido por la NSA mucho antes de 1970, cuando el DES se convirtió en un estándar. Es inocente creer que la NSA no haya desarrollado técnicas de criptoanálisis que pudieran ser aplicadas contra el DES, pero esos son sólo rumores.

Winn Schwartau escribió que la NSA construyó una máquina para atacar el DES a inicios de 1980. El rumor es que la NSA puede romper el DES en un tiempo de 3 a 15 minutos, dependiendo de cuanto pre-procesamiento puedan realizar, y estas máquinas costarían alrededor de $50,000 cada una.

La recomendación es utilizar las cajas de construcción de cajas de sustitución dependientes de llaves dada por Biham. Son fáciles de implementar en software y en chips. Esto incrementa la resistencia del algoritmo ante un ataque de fuerza bruta, hace que el análisis lineal y diferencial sea más difícil y le da a la NSA algo al menos tan fuerte como el DES, pero diferente, por lo cual preocuparse.

4.4 International Data Encryption Algorithm (IDEA)

El algoritmo IDEA es bastante más joven que el DES, pues data de 1992. Fue propuesto por Xuejia Lai y James Massey[59].

Trabaja con bloques de 64 bits de longitud y emplea una llave de 128 bits. El diseño del algoritmo de cifrado está basado en el concepto de "mezcla de operaciones de diferentes grupos algebraicos". El algoritmo fue desarrollado para incrementar la seguridad ante los criptoanálisis diferenciales.

Algunos expertos lo calificaron en su momento como el mejor y más seguro algoritmo disponible al público.[60]

Como en el caso del DES, se usa el mismo algoritmo tanto para cifrar como para descifrar.

IDEA esta considerado como un algoritmo inmune al criptoanálisis diferencial, tampoco existen ataques exitosos utilizando el criptoanálisis lineal, no se le conoce ninguna debilidad algebraica, además, la longitud de su llave hace imposible en la práctica un ataque por fuerza bruta. El criptoanálisis más significativo hacia el algoritmo lo hizo Daemen, quien descubrió una lista de 2^{51} llaves débiles, pero, considerando que existen 2^{128} posibles llaves, este resultado no tiene un impacto significativo[61].

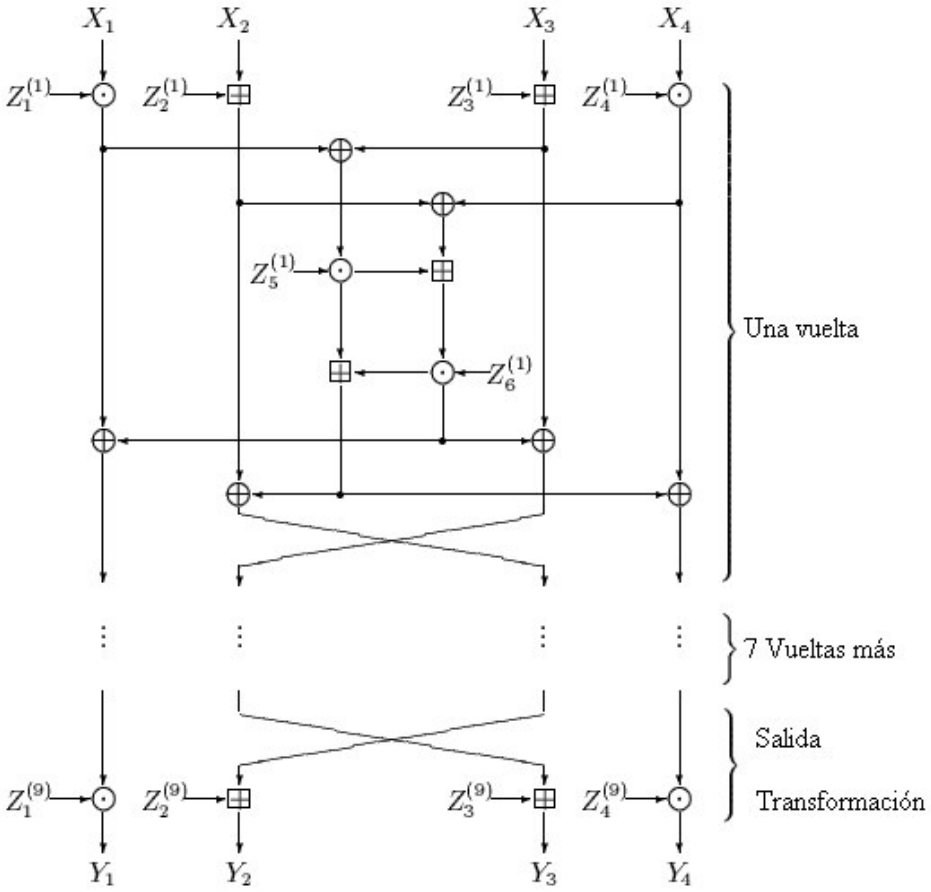
---

[59] Lai, X., Massey, J. *A Proposal for a New Block Encryption Standard*. Reprint of pp. 389-404 in Advances in Cryptology-EUROCRYPT'90 Proceedings, LNCS 473, Springer-Verlag, 1991.
[60] Schneier, B. Applied Cryptography. Pp. 319.
[61] http://www.rsasecurity.com/rsalabs/node.asp?id=2254

Como ocurre con todos los algoritmos simétricos de cifrado por bloques, IDEA se basa en los conceptos de confusión y difusión, haciendo uso de las siguientes operaciones elementales:

- XOR
- Suma módulo 2^{16}
- Producto módulo $2^{16} + 1$.

El algoritmo IDEA consta de ocho vueltas, después de las cuales se ejecuta una transformación. Esta última se toma en cuenta como una media vuelta extra. La primera vuelta completa y la transformación de salida son mostradas en la figura 4.14 Las dos multiplicaciones y las dos sumas mostradas en la figura son llamadas estructura MA, lo que hace referencia a la multiplicación y adición. La gestión de llaves toma una llave de entrada de 128 bits y retorna 52 sub-llaves, de 16 bits cada una, utilizando 6 para cada una de las 8 vueltas y 4 más para la transformación de salida.



"IDEA tiene un pequeñísimo subconjunto de llaves que pueden dar ciertas ventajas a un criptoanalista, pero la probabilidad de encontrarnos con una de ellas es de 1 entre 296, por lo que no representan un peligro real." Lucena, M. Op. Cit. pp. 90.
Schneier va más allá al afirmar que es fácil modificar IDEA de forma que carezca de llaves débiles, pues basta aplicar una función XOR entre cada sub-llave y 0x0DAE, donde x representa cualquier número. Schneier, B. Applied Cryptography. Pp. 323.

X_i : Sub bloque de texto claro de 16 bits
Y_i : Sub bloque de texto claro de 16 bits
$Z_i^{(r)}$: Sub bloque de llaves de 16 bits
$\oplus$: XOR sobre los sub bloques de 16 bits
$\boxplus$: Adición módulo 2^{16} de enteros de 16 bits
$\odot$: Multiplicación módulo $2^{16}+1$ de enteros de 16 bits con el sub bloque cero correspondiente a 2^{16}

Figura 4.20 Proceso de cifrado del algoritmo IDEA

4.4.1 Proceso de cifrado

Se divide el bloque *X* a codificar, de 64 bits, en cuatro partes X_1, X_2, X_3 y X_4 de 16 bits cada una. Se denomina *Zi* a cada una de las 52 sub-llaves de 16 bits que se necesitan. Las operaciones que se llevan a cabo en cada vuelta son las siguientes:

1. Multiplicar X_1 por Z_1
2. Sumar X_2 con Z_2
3. Sumar X_3 con Z_3.
4. Multiplicar X_4 por Z_4.
5. Hacer un XOR entre los resultados del paso 1 y el paso 3.
6. Hacer un XOR entre los resultados del paso 2 y el paso 4.
7. Multiplicar el resultado del paso 5 por Z_5.
8. Sumar los resultados de los pasos 6 y 7.
9. Multiplicar el resultado del paso 8 por Z_6.
10. Sumar los resultados de los pasos 7 y 9.
11. Hacer una operación XOR entre los resultados de los pasos 1 y 9.
12. Hacer una operación XOR entre los resultados de los pasos 3 y 9.
13. Hacer una operación XOR entre los resultados de los pasos 2 y 10.
14. Hacer una operación XOR entre los resultados de los pasos 4 y 10.

La salida de cada iteración serán los cuatro sub-bloques obtenidos en los pasos 11, 12, 13 y 14, que serán la entrada del siguiente ciclo, en el que se emplearán las siguientes seis sub-llaves, hasta un total de 48. Al final de todo el proceso, se intercambian los dos bloques centrales, con esto se deshace el intercambio que se llevó a cabo en los pasos 12 y 13.

Después de la octava iteración, se realiza la siguiente transformación:

1. Multiplicar X_1 por Z_{49}
2. Sumar X_2 con Z_{50}
3. Sumar X_3 con Z_{51}
4. Multiplicar X_4 por Z_{52}

Las primeras ocho sub-llaves se calculan dividiendo la llave de entrada en bloques de 16 bits. Las siguientes ocho se calculan rotando la llave de entrada 25 bits a la izquierda y volviendo a dividirla, y así sucesivamente.

Las sub-llaves necesarias para descifrar se obtienen cambiando de orden las Zi y calculando sus inversas para la suma o la multiplicación. Puesto que $2^{16} + 1$ es un número primo, nunca se podrá

obtener cero como producto de dos números, por lo que no se necesita representar dicho valor. Cuando se estén calculando productos, se utilizará el cero para expresar el número 2^{16} - un uno seguido de 16 ceros. Esta representación es coherente, puesto que los registros que se emplean internamente en el algoritmo poseen únicamente 16 bits.

Pese a las cualidades de este algoritmo, no llegó a reemplazar al DES como estándar en parte porque está patentado y se requiere una licencia para usarlo en aplicaciones comerciales y, por otra parte, por que aun se esperaba conocer su evolución ante los desarrollos en criptoanálisis posteriores a su surgimiento. Su fama se debe primordialmente a que es parte del PGP (Pretty Good Privacy), que se describe con más detalle en apartados posteriores.[62]

4.5 MADRYGA

W.E. Madryga propuso este algoritmo de cifrado de bloques en 1984. Es eficiente en cuanto al software, no requiere permutaciones complicadas y todas sus operaciones se realizan sobre bytes.

Sus objetivos de diseño son una reiteración de valores:

1. El texto claro no puede ser extraído del criptograma sin el uso de la llave (esto significa precisamente que el algoritmo es seguro).
2. El número de operaciones requeridas para determinar la llave a partir de una muestra de texto claro y del criptograma habrá de ser estadísticamente equiparable al producto de las operaciones de un cifrado tantas veces como posibles llaves haya.
3. El conocimiento del algoritmo no debería afectar la fortaleza del cifrado, permaneciendo toda la seguridad en la llave.
4. Un cambio de un bit en la llave produciría un cambio drástico en el criptograma usando el mismo texto claro, y de igual forma un cambio de un bit en el texto claro produciría un cambio drástico en el criptograma usando la misma llave (es decir que en cualquiera de los casos se desencadenaría el efecto avalancha).
5. El algoritmo contendría una combinación no conmutativa de sustituciones y permutaciones.
6. El algoritmo incluiría sustituciones y permutaciones tanto en el control de la entrada de datos como en la llave.
7. Los grupos de bits de redundancia estarían totalmente oscurecidos dentro del criptograma.
8. La longitud del texto claro y el cifrado serían iguales.
9. No habría relación entre posibles claves y efectos en el texto cifrado.
10. Cualquier llave posible produciría un cifrado fuerte, es decir que no habrían llaves débiles.
11. La longitud de la llave y el texto deberían ser ajustables para adecuarse a requerimientos de seguridad variables.
12. El algoritmo sería eficientemente implementable en software sobre *mainframes*, minicomputadoras y equipo basado en lógica discreta. (De hecho, este algoritmo recurre únicamente a funciones XOR y desplazamientos de bits).

[62] Schneier, B. Applied Cryptography. Pp. 320

4.5.1 Descripción del algoritmo

Madryga cuenta con dos ciclos anidados. La vuelta externa se repite ocho veces (aunque esto puede incrementarse por requerimientos de seguridad) y consta de una aplicación del ciclo interno al texto claro. El ciclo interno transforma el texto claro en texto cifrado y repite una vez cada bloque de ocho bits es decir cada byte del texto claro. De esta forma, el algoritmo recorre todo el texto claro ocho veces sucesivas.

Se opera una iteración del ciclo interno sobre una ventana de 3 bytes de datos, llamada *cuadro* o *trama de trabajo.* Lo anterior se ilustra en la Figura 4.15. Esta ventana avanza un byte en cada iteración. Vale aclarar que los datos son considerados circulares o rotativos cuando se trata de los últimos dos bytes. Los dos primeros bytes del cuadro de trabajo se rotan juntos un número variable de posiciones, mientras que al último se aplica una operación XOR con algunos bits de la llave. A medida que avanza el cuadro de trabajo, todos los bytes van siendo sucesivamente operados de la forma antes mencionada. Las rotaciones sucesivas traslapan los resultados de una operación XOR y rotación previas y el dato de la XOR es empleado para ejercer influencia sobre la rotación. Esto hace que todo el proceso sea reversible.

Debido a que cada byte de datos ejerce influencia sobre los dos que están a su izquierda y uno a su derecha, después de ocho veces que se ejecuta este procedimiento, cada byte del texto cifrado depende de los dieciséis bytes a su izquierda y los ocho a su derecha.

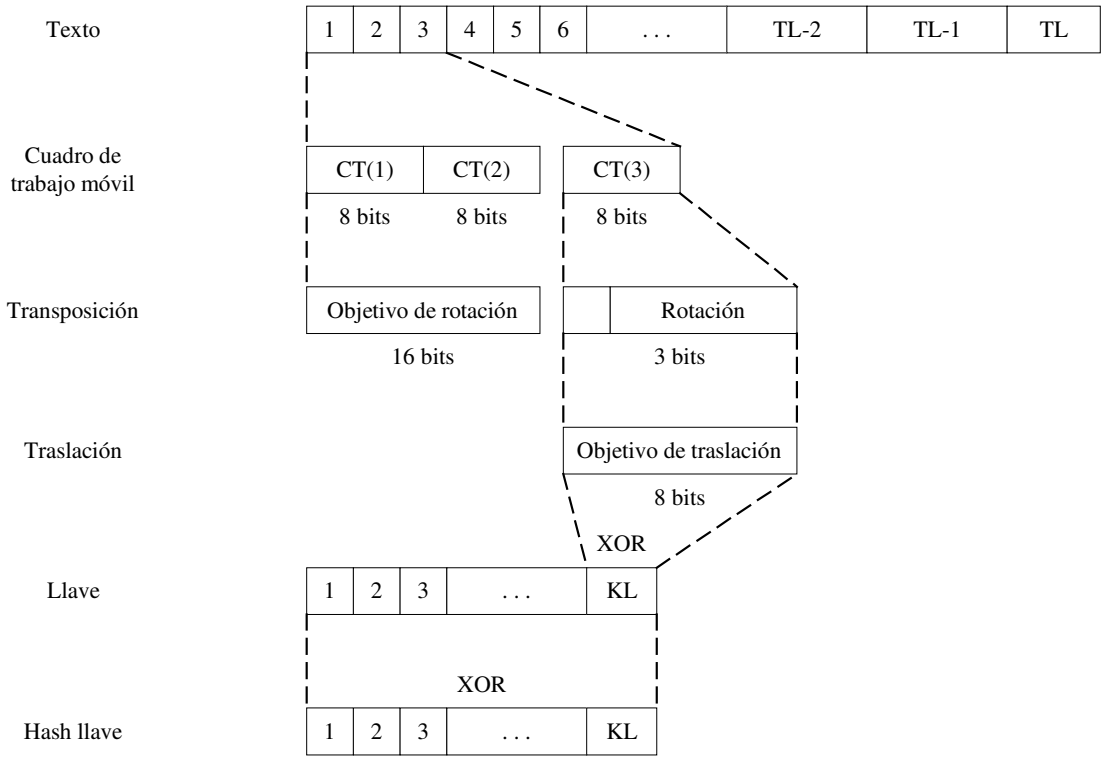


Figura 4.21 Una iteración de Madryga.

Durante el proceso de cifrado, cada iteración del ciclo interno inicia ubicando el cuadro de trabajo sobre el penúltimo byte de texto claro y avanza de forma circular hasta el antepenúltimo byte del texto claro. Primero se efectúa una operación XOR entre la llave entera y una constante aleatoria

y luego se le rota tres bits a la izquierda. Los tres bits menos significativos del byte menos significativo del cuadro de trabajo son guardados; ellos controlarán la rotación de los otros 2 bytes. Luego, se somete al byte menos significativo del cuadro de trabajo a una operación XOR con el byte menos significativo de la llave. Después, la concatenación de los dos bytes más significativos es rotada a la izquierda un número variable de bits (entre cero y siete). Finalmente, se desplaza el cuadro de trabajo 1 byte a la derecha y se repite todo el proceso.

El punto de la constante aleatoria es convertir la llave en una secuencia seudoaleatoria. La longitud de esta constante debe ser igual a la de la llave y debe ser la misma para todos aquellos que deseen comunicarse con otro. Para una llave de 64 bits, Madryga recomienda la constante `0X0F1E2D3C4B5A6978`.

El descifrado revierte el proceso. Cada iteración del ciclo interno comienza con el cuadro de trabajo en el antepenúltimo byte del texto cifrado y avanza en sentido contrario circularmente, pasando al penúltimo byte. Tanto la llave como los dos bytes de texto cifrado son desplazados hacia la derecha y se efectúa una operación XOR antes de las rotaciones.

Schneier no recomienda el uso de este algoritmo debido a una serie de problemas que fueron detectados tanto por investigadores Queensland University of Technology (QUT) como por Eli Biham[63]. Schneier dice que si bien las dificultades detectadas no son condenables en si mismas, el conjunto de ellas no permite confiar en el algoritmo.

Otros autores coinciden con Schneier y llegan a afirmar que para hacer ataques diferenciales a este algoritmo basta cantidades bastante pequeñas de datos y la ruptura se obtiene de una forma bastante rápida. Lo anterior revela que Madryga es muy débil en comparación con otros algoritmos de cifrado[64].

---

[63] Según refiere Schneier, el equipo de QUT examinó una serie de algoritmos de cifrado de bloques entre los que se encontraba Madryga y observaron que en este último no exhibía el efecto avalancha entre texto claro y texto cifrado. Adicionalmente, cita las siguientes observaciones emitidas por Biham:
"El algoritmo solo recurre a operaciones lineales (rotaciones y XOR), que son escasamente modificadas dependiendo de los datos.
No hay nada como la fortaleza de las cajas de sustitución del DES.
La paridad de todos los bits en el texto claro y el cifrado es constante, dependiendo únicamente de la llave. Esto lleva a que si se tiene un texto claro y su correspondiente cifrado, es posible predecir la paridad de ambos". Schneier, B. Applied Cryptography. Pp. 306.

[64] El problema fundamental es la falla en la deseada propiedad de avalancha, pues un bit provocará avalancha solo si es tal la condición que modifica la rotación; ya que solo 3 de los 6 bits son usados en la rotación, un cambio de bit tiene una oportunidad de 5/8 de no causar alteraciones durante un paso particular del ciclo interno, así como durante la vuelta completa. En esas condiciones, es obvio que resulta poco probable que se desencadene una avalancha. Adicionalmente, se coincide con Biham en los señalamientos sobre el problema de paridad de este algoritmo. Shirriff, K. *Differential Cryptanalysis of Madryga*. Sun Microsystems Labs. Draft Version. October 19, 1995.

4.6 SKIPJACK

Este algoritmo de cifrado fue desarrollado por la NSA e incorporado en los chips CLIPPER y CAPSTONE. Estos fueron propuestos por los Estados Unidos para la aplicación de criptografía con el polémico método consistente en que los usuarios depositen sus claves secretas en diferentes agencias del gobierno. Tanto los chips como el algoritmo están clasificados por el gobierno de los Estados Unidos como secretos del más alto nivel. Por esta razón, la información conocida acerca de este método de cifrado es bastante reducida. Entre ella se encuentran, por ejemplo, los siguientes datos:

- Es un cifrado de bloque iterativo
- Cifra bloques de información de 64 bits
- Utiliza una clave de cifrado de 80 bits
- Se puede utilizar con los modos de cifrado ECB, CBC, OFB y CFB
- Consta de 32 vueltas elementales de cifrado

El método debe ser seguro puesto que la NSA tiene la intención de utilizarlo en el cifrado de su DMS (Defense Messaging System). Un panel de expertos fue informado de que pasarán más de 36 años hasta que la dificultad de romper el SKIPJACK mediante un ataque basado en fuerza bruta sea equivalente al esfuerzo requerido hoy en día para romper DES utilizando este mismo tipo de ataque.

El cifrador SKIPJACK esta patentado pero la patente es secreta.[65]

[65] Pastor, J., Sarasa, M. Op. Cit. Pp. 124

4.7 BLOWFISH

Blowfish es un algoritmo de cifrado de bloques propuesto por Bruce Schneier[66]. Se trata de una red de Feistel que itera una función de cifrado dieciséis veces. Los bloques son de 64 bits y la llave puede llegar a alcanzar cualquier longitud hasta 448 bits. Si bien requiere una fase de inicialización relativamente compleja antes de efectuar cualquier cifrado, es muy eficiente cuando se implementa sobre grandes procesadores.

Según refiere Schneier[67], diseñó este algoritmo siguiendo los siguientes criterios:

1. Rápido. Blowfish cifra datos en microprocesadores de 32 bits a 26 ciclos de reloj por byte.
2. Compacto. Blowfish puede ser ejecutado en menos de 5Kbytes de memoria.
3. Simple. Blowfish emplea únicamente operaciones simples: adición, XOR y tablas de búsqueda (*lookups tables*) sobre operandos de 32 bits. Es fácil analizar su diseño y ello lo vuelve resistente a errores de implementación.
4. Ajustable en términos de seguridad. La longitud de la llave de Blowfish es variable y puede llegar a tener hasta 448 bits.

Este algoritmo está optimizado para aplicaciones donde la llave no cambia con mucha frecuencia, como en enlaces de comunicación o en cifradores automáticos de archivos. Es significativamente más rápido que el DES cuando se implemente sobre procesadores de 32 bits con cache de datos grande. No es recomendable su aplicación en conmutación de paquetes (por la necesidad de mayor frecuencia en los cambios de llave) o como función hash de sentido único. Sus requerimientos de hardware lo hacen inconveniente en tarjetas inteligentes.

4.7.1 El funcionamiento de Blowfish

El algoritmo consta de dos partes: la expansión de llave y el cifrado de datos. La expansión de llave convierte una llave de hasta 448 bits en varios arreglos de sub-llaves totalizando 4168 bytes. El cifrado de datos consiste en la iteración de una función simple dieciséis veces. Cada iteración es una permutación dependiente de la llave y una sustitución dependiente de la llave y de los datos. Las operaciones se limitan a sumas y XOR sobre palabras de 32 bits. Las únicas operaciones adicionales son cuatro tablas de búsqueda de arreglos indexados por vuelta.

Blowfish utiliza una gran cantidad de sub-llaves. Ellas pueden ser calculadas con anticipación, como paso previo a cualquier proceso de cifrado o descifrado.

El arreglo *P* consta de 18 sub-llaves de 32 bits: $P_1, P_2, \ldots, P_{18}$.

Hay cuatro cajas de sustitución de 32 bits, cada una de ellas con 256 entradas:

$$S_{1,0}, S_{1,1}, \ldots, S_{1,255}$$
$$S_{2,0}, S_{2,1}, \ldots, S_{2,255}$$
$$S_{3,0}, S_{3,1}, \ldots, S_{3,255}$$
$$S_{4,0}, S_{4,1}, \ldots, S_{4,255}$$

[66] Schneier, B. *Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish).* Fast Software Encryption, Cambridge Security Workshop Proceedings (December 1993), Springer-Verlag, 1994, Pp. 191-204.
[67] Schneier, B. Applied Cryptography. Pp. 336.

Como se mencionó al principio de este apartado, Blowfish es una red de Feistel de 16 vueltas. La entrada es un elemento de datos de 64 bits al que por simplicidad llamaremos x. El procedimiento de cifrado es el siguiente:

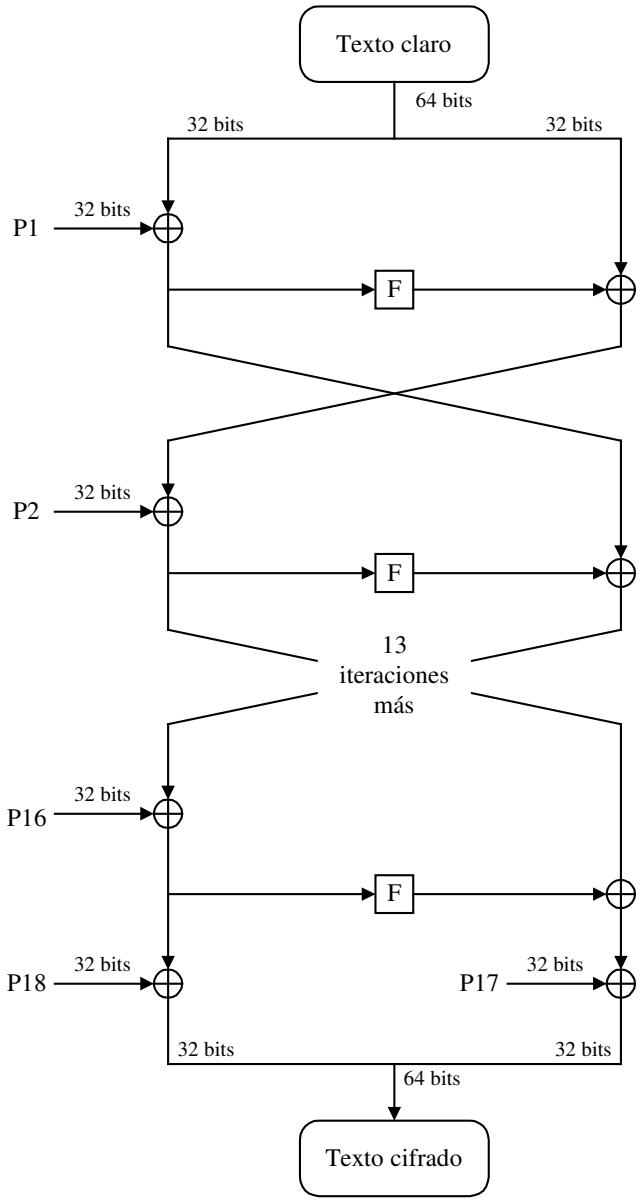


Figura 4.22 Estructura del algoritmo Blowfish

Lo presentado en la Figura 4.22 puede expresarse de la manera siguiente:

Dividir x en mitades de 32 bits: $x_L,\ x_R$
Para i=1 hasta 16:
$x_L = x_L$ XOR P_i
$x_R = F(x_L)$ XOR x_R
Intercambiar las posiciones de x_L y x_R
Intercambiar las posiciones de x_L y x_R (deshacer el último intercambio)

$x_R = x_R$ XOR P_{17}
$x_L = x_L$ XOR P_{18}
Recombinar x_L y x_R

En cuanto a la función *F*, ésta se implementa de la forma siguiente:

Dividir x_L en cuatro fragmentos de ocho bits:
a, *b*, *c* y *d* $F(x_L) = ((S_{1,a} + S_{2,b} \bmod 2^{32})$ XOR $S_{3,c}) + S_{4,d}$

Es proceso de descifrado es exactamente similar, salvo que $P_1, P_2, \ldots, P_{18}$.se emplean en orden inverso.

El contenido de la función *F* se ilustra en la Figura 4.23.

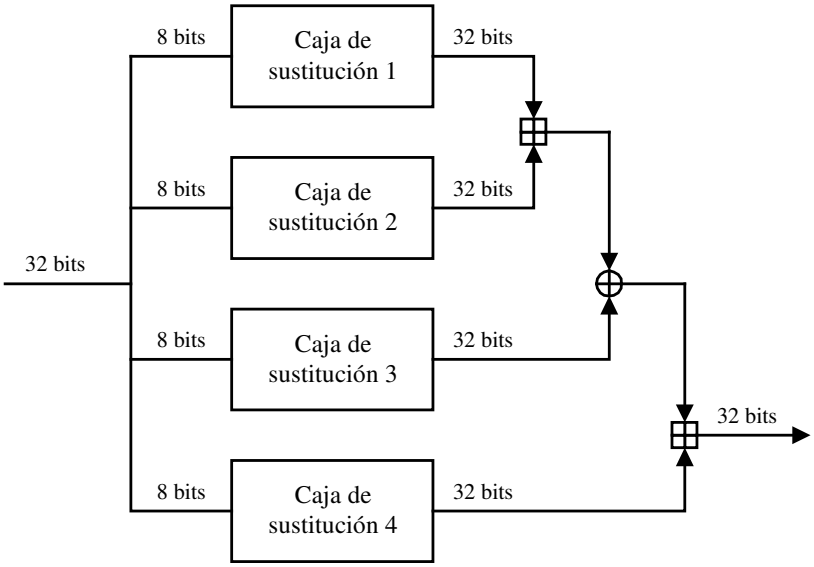


Figura 4.23 La función *F*

Las sub-llaves se calculan de acuerdo al método siguiente:

1. Inicializar primero el arreglo *P* y luego las cuatro cajas de sustitución en orden con una cadena fija. Esta cadena consiste en los dígitos hexadecimales de π.
2. Efectuar una operación XOR entre P_1 y los primeros 32 bits de la llave, efectuar otra operación XOR entre P_2 y la otra mitad de la llave, y seguir así sucesivamente hasta llegar a P_{18}. Iterar mediante los dígitos de la llave hasta que se hayan ejecutado las correspondientes operaciones XOR entre los bits de la llave y todo el arreglo *P*.
3. Cifrar una cadena de ceros con el algoritmo, usando las sub-llaves descritas en los dos pasos anteriores.
4. Reemplazar P_1 y P_2 con la salida del paso 3.
5. Cifrar la salida del paso 3 usando el algoritmo con las llaves modificadas.
6. Reemplazar P_3 y P_4 con la salida del paso 5.
7. Continuar el proceso, reemplazando todos los elementos del arreglo *P* y luego todas las cajas de sustitución en orden, con la salida continuamente cambiante del algoritmo.

Se requieren en total 521 iteraciones para generar todas las sub-llaves. Estas pueden ser almacenadas en aplicaciones, pues no es necesario ejecutar toda esta etapa del proceso múltiples veces.

4.7.2 La seguridad de Blowfish

Serge Vauldenay examinó este algoritmo. Este investigador indica que son viables los ataques por criptoanálisis diferencial contra Blowfish sobre un número reducido de vueltas o con la parte de información que describe la función *F*. El segundo caso parece ser equivalente a un análisis efectuado por Lee, Heys y Tavares sobre otro algoritmo, el CAST; sin embargo, en el caso de CAST, las cajas de sustitución son diseñadas de manera que se evite cualquier ataque, mientras que en Blowfish son generadas aleatoriamente. Vaudenay investigó las debilidades de las cajas de sustitución sobre la base de colisiones y demostró que hay llaves débiles en Blowfish que habilitan una disminución significativa de la complejidad de los ataques y que incluso es posible detectar llaves débiles usando 2^{22} textos claros selectos (con ocho iteraciones).[68]

4.8 RIJNDAEL

El algoritmo Rijndael fue desarrollado por Vincent Rijmen y Joan Daemen, dos investigadores belgas. El primero nació en 1965 y se graduó de Ingeniero civil electromecánico y luego se doctoró en Criptografía. El segundo nació en 1970, se graduó de la carrera de ingeniería electrónica en la Universidad Católica de Lovaina, de donde posteriormente se doctoró realizando una tesis sobre Criptoanálisis y diseño de cifrados de bloque iterativos.

Rijmen y Daemen iniciaron el desarrollo de un algoritmo al que denominaron *Square*[69]. Su principal diferencia con los algoritmos existentes es que trabaja con llaves y bloques de 128 bits. Este algoritmo se hizo público en 1997. Coincidiendo con la fecha, el NIST anunció el concurso para la selección del próximo AES, *Advanced Encription Standard*. La condición de estándar pertenecía al algoritmo DES desde la década de los 70. Como medida temporal mientras no se seleccionaba el nuevo AES, el NIST recomendó el uso del Triple DES para garantizar la privacidad de los documentos.

Uno de los requisitos que los algoritmos debían cumplir para participar en el concurso de selección del AES era que debían trabajar con llaves de 128, 192 y 156 bits y longitudes de bloque de al menos 128 bits. Rijmen y Daemen iniciaron la modificación de Square para adaptarlo a las condiciones del concurso y en junio de 1998 remitieron al NIST un nuevo algoritmo, descendiente de Square, al que llamaron Rijndael como una alusión a sus apellidos (*Rij*me*n*&*Dae*men).[70]
En agosto de 1998 finalizó el plazo para la presentación de algoritmos, se obtuvieron 15 propuestas, de las cuales tres, Rijndael, Crypton y Twofish, estaban basadas en la estructura de Square. Los algoritmos fueron analizados por un grupo de expertos designados por el NIST que

[68] Vauldenay, S. *On the Weak Keys of Blowfish*. École Normale Supérieure – DMI.Laboratoire d'Informatique. Paris. S/f.
[69] Algoritmo de cifrado de bloques iterativo con longitud de bloques y llave de 128 bits. Para mayores detalles, consultar Daemen, J., Knudsen, L., Rijmen, V. *The Block Cipher Square*. Fast Software Encryption, LNCS 1267, E. Biham, Ed., Springer-Verlag, 1997, pp. 149-165. Also available as http://www.esat.kuleuven.ac.be/rijmen/square/fse.ps.gz.
[70] La propuesta que remitieron al NIST puede ser consultada en el documento Daemen, J., Rijmen, V. *AES Proposal: Rijndael*. Document version 2, Date: 03/09/99.

valoraron aspectos tales como: robustez, estructura, capacidad para ser implementado en software y hardware, etc.

En agosto de 1999 se dieron a conocer los cinco finalistas: Twofish, Mars, Serpent, RC6 y Rijndael; este último se destacó frente a sus competidores, ya que tanto la longitud de su llave, como el tamaño de los bloques que cifra podía ser de 128, 192 ó 256 bits. Fue así como el 2 de octubre de 2000 se proclamó a Rijndael como ganador del concurso y nuevo AES.

Sobre las razones por las que se le eligió frente a los otros candidatos, el NIST declaró: *"Tomando todo en consideración, la combinación de seguridad, rendimiento, eficiencia, facilidad de implementación y flexibilidad lo hacían la elección adecuada para el AES. Específicamente, Rijndael consistentemente obtiene muy buen rendimiento tanto en hardware como en software en una amplia variedad de entornos de computación tanto usado en modos feedback como no feedback. Su preparación de llaves es excelente y la agilidad de las mismas muy buena. Rijndael requiere muy poca memoria lo que lo hace excelente para entornos con espacio restringido demostrando aquí también su excelente rendimiento. Las operaciones de Rijndael están entre las más sencillas de defender contra ataques por análisis temporal (timing attack) y criptoanálisis diferencial de potencia (power attack)".*

Por otro lado parece que se puede proporcionar defensa contra los ataques citados sin afectar significativamente el rendimiento de Rijndael. El algoritmo se ha diseñado con flexibilidad en términos de los tamaños de bloque y de llave. Finalmente, la estructura interna de las rondas de Rijndael parece tener un buen potencial para beneficiarse de los procesos en paralelo.

El NIST afirmó también que en términos de seguridad los cinco finalistas eran adecuados, pero Rijndael era el que mejor combinaba el resto de características deseables.

4.8.1 Especificaciones del algoritmo

El algoritmo Rijndael es un cifrador de bloques, de carácter iterativo, es decir, realiza varias vueltas de cifrado sobre el mismo conjunto de bits. Además, permite especificar un tamaño de llave y de bloque de 128, 192 ó 156 bits, y lo que es más importante, combinarlos entre sí de diferentes maneras, lo que da como resultado nueve asociaciones entre el tamaño de la llave y del bloque.

4.8.1.1 Los Estados, las llaves y el número de vueltas

Las diferentes transformaciones de los resultados intermedios son llamadas Estados. Cada uno de los estados intermedios del algoritmo puede ser representado como una matriz de bytes. Esta matriz tiene 4 filas y el número de columnas, denotado por N_b, es variable, siendo igual al tamaño del bloque dividido por 32, es decir, para un tamaño de bloque de 128 bits le corresponden 4 columnas (128/32), para un tamaño de 192 bits le corresponden 6 columnas (192/32) y para 256 bits le corresponden 8 columnas.

Las llaves de los estados intermedios también pueden ser representados de manera similar mediante una matriz de cuatro filas y un número variable de columnas, que se denota por N_k y que se obtiene al dividir el tamaño de la llave por 32, esto es, 4, 6, 7 u 8 columnas para tamaños de llave de 128, 192 y 256 bits respectivamente.

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

Estado intermedio con tamaño de bloque de 128 bits

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

Representación de una llave de 128 bits

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

Estado intermedio con tamaño de bloque de 192 bits

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$	$k_{0,4}$	$k_{0,5}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$	$k_{1,4}$	$k_{1,5}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$	$k_{2,4}$	$k_{2,5}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$	$k_{3,4}$	$k_{3,5}$

Representación de una llave de 192 bits

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$	$a_{0,6}$	$a_{0,7}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$a_{1,6}$	$a_{1,7}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$	$a_{2,6}$	$a_{2,7}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$	$a_{3,6}$	$a_{3,7}$

Estado intermedio con tamaño de bloque de 256 bits

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$	$k_{0,4}$	$k_{0,5}$	$k_{0,6}$	$k_{0,7}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$	$k_{1,4}$	$k_{1,5}$	$k_{1,6}$	$k_{1,7}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$	$k_{2,4}$	$k_{2,5}$	$k_{2,6}$	$k_{2,7}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$	$k_{3,4}$	$k_{3,5}$	$k_{3,6}$	$k_{3,7}$

Representación de una llave de 256bits

Tabla 4.9 Bloques y llaves de Rijndael

En determinadas ocasiones las matrices mostradas anteriormente se pueden considerar como un único vector unidimensional de palabras de 4 bytes, donde cada palabra se corresponde con una de las columnas de la matriz. Estos vectores tendrían por tanto una longitud de 4, 6 y 8 palabras según el número de columnas de la matriz.

La entrada y salida de datos en Rijndael se realiza mediante un vector unidimensional de bytes de 8 bits por byte. Estos bytes están numerados con valores que parten desde el 0 hasta $(4*N_b)$-1, es decir, para $N_b = 4$ el número de bytes sería 16 y estarían numerados desde el 0 hasta el 15. En la tabla x se pueden apreciar las relaciones entre el tamaño del bloque, el valor de N_b y el rango de índices correspondientes del vector entrada-salida.

Número de bits	Número de bytes	Nb	Tamaño del vector de entrada	Rango de índices
128	16	4	16	0..15
192	24	6	24	0..23
256	32	8	32	0..31

Tabla 4.10 Tamaño de bits y el valor Nb

Una vez se tiene el vector de entrada completo, se debe volcar el contenido en la matriz correspondiente. El modo de realizar esta operación es el siguiente: se leerá el vector secuencialmente desde la posición 0 y se irá rellenando la matriz por columnas.

Ejemplo:
Volcado de información para el caso de un bloque de 128 bits. El vector de entrada es de 16 bytes numerados del 0 al 15. V es el vector de entrada y sus componentes se denotan por V_j.

V_o	V_1	V_2	V_3	V_4	V_5	V_6	V_7	**V_8**	**V_9**	**V_{10}**	**V_{11}**	V_{12}	V_{13}	V_{14}	V_{15}

V_o	V_4	**V_8**	V_{12}
V_1	V_5	**V_9**	V_{13}
V_2	V_6	**V_{10}**	V_{14}
V_3	V_7	**V_{11}**	V_{15}

Tabla 4.11 Volcado de información desde un vector de entrada a la matriz

Para volcar el contenido de la matriz en un vector de salida basta con aplicar el mismo proceso en orden inverso: se lee la matriz por columnas, empezando en la columna 0 y rellenando el vector desde la posición 0 hasta el final.

4.8.1.2 Número de vueltas

El número de vueltas que tendrá el algoritmo es un parámetro variable que dependerá de los valores N_b y N_k, y estos a su vez dependen del tamaño del bloque y del tamaño de llave respectivamente. El número de vueltas se denota N_r.

N_r	**$N_b = 4$**	**$N_b = 6$**	**$N_b = 8$**
$N_b = 4$	10	12	14
$N_b = 6$	12	12	14
$N_b = 8$	14	14	14

Tabla 4.12 Número de vueltas del algoritmo según el valor de Nb y Nk

El número de vueltas del algoritmo se encuentra en proporción al tamaño del bloque y la llave.

4.8.1.3 Operaciones en cada vuelta

El algoritmo se compone de una vuelta inicial (RoundKey), y *r* vueltas estándar, *r* puede tomar el valor de 10, 12 ó 14 dependiendo de la longitud del bloque y la llave. Las primeras *r*-1 vueltas son similares y consisten en cuatro transformaciones llamadas:

- ByteSub
- ShiftRow
- MixColumn
- RoundKey

La última vuelta carece de la operación MixColum.

Todo el proceso de cifrado se muestra en el siguiente diagrama:

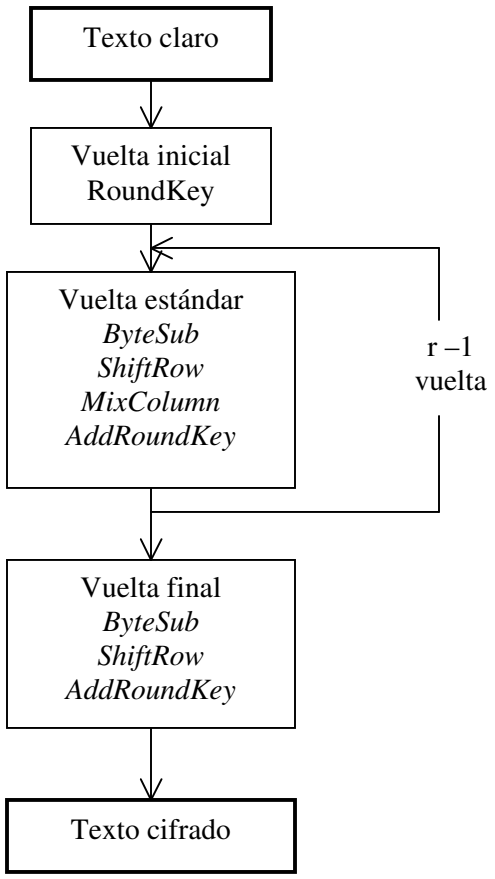


Figura 4.24 Procedimiento de cifrado empleado por el algoritmo Rijndael

4.8.1.4 Transformación BytesSub

Se utilizan cajas de sustitución, se trata de una sustitución a nivel de byte de caracter no lineal. Se aplica sobre todos y cada uno de los bytes de un estado de manera independiente.

La sustitución que realiza es inversible y se compone de dos pasos:

1. Se aplica la función inversa para la multiplicación en $GF(2^8)$[71]. Cada elemento se sustituye por su valor inverso, salvo el valor 00, que se sustituye por sí mismo ya que carece de inverso para la multiplicación con coeficientes pertenecientes a $GF(2^8)$.

[71] Campos de Galois

2. 2. Se aplica la siguiente transformación

$$\begin{vmatrix} Y_o \\ Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \\ Y_5 \\ Y_6 \\ Y_7 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{vmatrix} \cdot \begin{vmatrix} X_o \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ Y_7 \end{vmatrix} + \begin{vmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{vmatrix}$$

Donde cada X_j representa el bit j del valor del byte al que se le va a aplicar la transformación y cuyo valor final se representa por Y_j.

Características del proceso de sustitución

1. Inversible
2. Minimiza el total de números que no sean combinaciones lineales de los bits de entrada ni sean bits de salida
3. Minimiza el mayor de los números no triviales de la tabla XOR
4. Expresiones algebraicas complejas con coeficientes pertenecientes a los campos de Galois, GF(2^8).
5. Sencillo de describir

El diseño modular del algoritmo permite la posibilidad de que si en un futuro se encontrase una caja de sustitución más afín a las necesidades del entorno, se podría cambiar por una nueva versión sin tener que modificar en absoluto el resto del algoritmo y de forma rápida.

4.8.1.5 La transformación ShiftRow

Esta transformación consiste en el desplazamiento, de forma cíclica, a la izquierda de los bytes de cada una de las filas de la matriz representante de un estado.
Los bytes de la fila 0 de la matriz no se desplazan. Los bytes de las filas 1, 2 y 3 se desplazan C_1, C_2 y C_3 posiciones respectivamente a la izquierda. Estos valores de desplazamiento dependen del valor de N_b, es decir del número de columnas de la matriz, o lo que es lo mismo, del tamaño del bloque. En la siguiente tabla se muestran dichos valores.

N_b	C_1	C_2	C_3
4	1	2	3
6	1	2	3
8	1	3	4

Tabla 4.13 Valor de los desplazamientos C1, C2 y C3 según Nb.

Para comprender mejor el funcionamiento de esta transformación se utilizará un ejemplo en el cual se transformará en un estado con $N_b = 4$. En este caso los desplazamientos serán de 1, 2 y 3 para C_1, C_2 y C_3 respectivamente. A la fila 0 no se le aplica la transformación, los bytes de la fila

1 se desplazarán 1 posición a la izquierda, los de la fila 2 se desplazarán 2 posiciones a la izquierda y los de la fila 3 se desplazarán 3 posiciones.

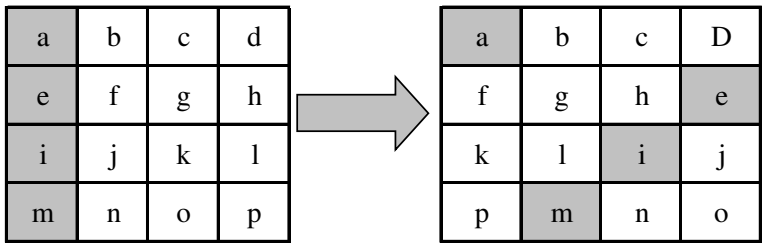


Figura 4.25 Ejemplo de desplazamiento de bytes C_1, C_2 y C_3

La operación inversa de esta transformación consiste simplemente en desplazar los bytes de las filas 1, 2 y 3 un total de C_1, C_2 y C_3 posiciones a la derecha.

Características de los desplazamientos

La posibilidad de elegir cualquier posible combinación se basa en los siguientes criterios:

1. Los cuatro desplazamientos deben ser diferentes y $C_0 = 0$
2. Resistencia frente a los ataques usando el método de criptoanálisis llamado diferenciales truncados *(truncated differentials*[72]*)*
3. Resistencia frente al ataque *Square*
4. Simplicidad

Para algunas combinaciones los ataques basados en truncated differentials pueden necesitar varias vueltas, aunque lo normal es que se emplee solamente una.

El número de vueltas necesarias para una ataque *Square* varía mucho en función de la combinación elegida, por lo que no se puede indicar un número concreto como valor medio de vueltas.

4.8.1.6 La transformación MixColumn

En esta transformación las columnas de la matriz representante del estado son consideradas como polinomios con coeficientes pertenecientes a $GF(2^8)$. Estos polinomios se multiplican modulo M(x), ($M(x) = X^4 + 1$), por un polinomio d(x) dado:

$d(x) = \text{‘03’}\ x^3 + \text{‘01’}x^2 + \text{‘01’}x + \text{‘02’}$ en hexadecimal
$d(x) = 11x^3 + 1x^2 + 1x + 10$ en binario
$d(x) = 3x^3 + 1x^2 + 1x + 2$ en decimal

[72] Ver Anexo dedicado a Criptoanálisis.

Esta operación puede ser escrita en forma de matriz, donde aj representa el byte j de la columna de la matriz estado y bj al nuevo byte tras la operación:

$$\begin{vmatrix} d_o \\ d_1 \\ d_2 \\ d_3 \end{vmatrix} = \begin{vmatrix} a0 & a3 & a2 & a1 \\ a1 & a0 & a3 & a2 \\ a2 & a1 & a9 & a3 \\ a3 & a2 & a1 & a0 \end{vmatrix} \cdot \begin{vmatrix} b_o \\ b_1 \\ b_2 \\ b_3 \end{vmatrix}$$

La inversa de esta operación consiste en multiplicar la columna transformada por el inverso del polinomio anterior d(x) para obtener la columna inicial.

Se denota a(x) al inverso del polinomio d(x), entonces:

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

Características del proceso de mezcla de columnas

Es una transformación interna en el conjunto de las palabras de 4 bytes. Para su elección fueron determinantes los siguientes criterios:

- Ser inversible
- Linealidad
- Poder de difusión
- Rapidez en procesadores de 8 bits
- Simetría
- Sencillez de descripción

La elección de la multiplicación polinómica módulo $x^4 + 1$ se decidió basándose en los criterios 2, 5 y 6. Los criterios 1, 3 y 4 impusieron restricciones a los coeficientes de las operaciones que se efectúan.

En el caso concreto del criterio 4, éste motivó que los coeficientes tuviesen valores reducidos, tales como ‘00’, ’01’, ’02’, ’03’. El valor ‘00’ implica que no se necesita realizar ninguna operación; con el valor ’01’ no se necesita realizar ninguna multiplicación.

El criterio 3 introduce restricciones mucho más complicadas sobre los coeficientes para conseguir su propósito.

4.8.1.7 La adición de las sub llaves - AddRoundKey

Esta operación consiste en una operación XOR entre los elementos de la matriz del estado y los elementos de la matriz de la sub-llave. Esta sub-llave es el resultado de aplicar el proceso llamado *Key schedule*. La matriz de la llave tiene el mismo número de columnas que la matriz de bloque (*Nb*).

La inversa de la operación es ella misma. Al aplicar la función XOR a la matriz ya transformada se obtiene de nuevo la matriz inicial.

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

$\oplus$

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$	$k_{0,4}$	$k_{0,5}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$	$k_{1,4}$	$k_{1,5}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$	$k_{2,4}$	$k_{2,5}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$	$k_{3,4}$	$k_{3,5}$

=

$b_{0,0}$	$b_{0,1}$	$b_{0,2}$	$b_{0,3}$	$b_{0,4}$	$b_{0,5}$
$b_{1,0}$	$b_{1,1}$	$b_{1,2}$	$b_{1,3}$	$b_{1,4}$	$b_{1,5}$
$b_{2,0}$	$b_{2,1}$	$b_{2,2}$	$b_{2,3}$	$b_{2,4}$	$b_{2,5}$
$b_{3,0}$	$b_{3,1}$	$b_{3,2}$	$b_{3,3}$	$b_{3,4}$	$b_{3,5}$

Tabla 4.14 Adición de la llave mediante la función XOR

4.8.1.8 Gestión de llaves

Mediante una función llamada *Key schedule* se obtienen las diferentes sub-llaves a partir de la llave principal de cifrado. Se compone a su vez de dos funciones, la expansión de llave, *KeyExpansion* y la selección de llave en cada vuelta, *Round KeySelection*. La función *Key schedule* se basa en lo siguiente: El número total de sub-llaves es igual al tamaño del bloque multiplicado por el número de vueltas más 1.

Número de sub-llaves = N_b * (N_r + 1)

Tamaño de bloque	Tamaño de la llave	Número de vueltas	Número de sub-llaves
128	4	10	1408
	6	12	1664
	8	14	1920
192	4	12	2496
	6	12	2496
	8	14	2880
256	4	14	3840
	6	14	3840
	8	14	3840

Tabla 4.15 Número de sub-llaves generadas

- La llave del cifrado se expande y pasa a denominarse *Llave Expandida*

- Las sub-llaves provienen de la *Llave Expandida*. Para obtenerlas se divide la llave expandida en fragmentos de tamaño *Nb*. El primero de estos fragmentos será la primera sub-llave; el segundo fragmento será la segunda sub-llave y así sucesivamente.

4.8.1.9 Expansión de la llave

La llave expandida proviene de la llave principal de cifrado. No existe la posibilidad de especificar una llave expandida concreta. Es un vector de palabras de 4 bytes y se denota por W[N_b*(N_r + 1)]. Esta llave expandida se puede crear en memoria usando un *buffer* para N_k palabras y así ahorrar trabajo de procesamiento en implementaciones donde la memoria RAM sea

escasa. Las primeras N_k palabras contienen la llave de cifrado principal. Las restantes palabras son definidas recursivamente en términos de palabras con índices cada vez más pequeños.

La función *KeyExpansion* depende del valor de N_k (número de columnas de la matriz representante de la llave o, lo que es lo mismo, número de palabras de 4 bytes que contienen la llave).

Características de la función de expansión de la llave

La función *KeyExpansion* especifica la obtención de las sub-llaves en función de la llave principal de cifrado. Su objetivo es proveer al algoritmo de resistencia frente a los siguientes tipos de ataques:

- Ataques donde parte de la llave principal de cifrado es conocida por el criptoanalista.
- Ataques donde la llave principal de cifrado sea conocida o pueda ser seleccionada.

La función fue seleccionada bajo los siguientes criterios:

- Puede emplear una transformación inversible.
- Velocidad en gran número de procesadores.
- Uso de constantes en las vueltas para eliminar simetrías.
- Difusión de las diferencias de la llave de cifrado en las sub-llaves.
- El conocimiento de parte de la llave principal de cifrado o de bits de alguna sub-llave no permite obtener el resto de los bits que componen las sub-llaves
- La no–linealidad de la función, impide determinar la totalidad de las diferencias de las sub-llaves partiendo únicamente de las diferencias de la llave principal de cifrado
- Sencilla de describir

4.8.1.10 Selección de llave en cada vuelta - Round Key Selection

Las sub-llaves proceden de la llave expandida. Para seleccionar la sub-llave *i* se seleccionan las palabras de la llave expandida comprendidas entre $W[N_b*i]$ y $W[N_b*(i+1)]$.

W_0	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}	W_{11}	W_{12}	W_{13}	W_{14}	.....

Sub-llave 0	Sub-llave 1	.....

Tabla 4.16 Llave expandida y selección de sub-llaves para $N_k = 4$ y $N_b = 6$

4.8.2 Aspectos de implementación

El algoritmo Rijndael es puede ser implementado de una manera eficiente en variedad de procesadores y hardware, por ejemplo, procesadores de 8 bits, típicos en las tarjetas inteligentes, y en los procesadores de 32 bits disponibles en la mayoría de las computadoras personales.

4.8.2.1 Procesadores de 8 bits

En este tipo de procesadores se puede programar con el algoritmo Rijndael simplemente implementando las correspondientes transformaciones. Para la implementación de la función SubByte se necesita una tabla de 256 bytes.

Las operaciones *SubByte*, *RoundShift* y *Round key addition* se pueden combinar y ejecutar secuencialmente en los bytes de un estado concreto. La operación *MixColumn* requiere una matriz multiplicativa con coeficientes pertenecientes a GF(2^8).

La implementación del algoritmo para que sea ejecutado en un solo ciclo requiere una gran cantidad de memoria RAM. Además, la mayoría de las aplicaciones, tales como una tarjeta de crédito o cajeros electrónicos, trabajan con un número reducido de bloques de datos, por lo que el aumento de la velocidad obtenido al implementar funciones en paralelo apenas compensa el gasto adicional de memoria.

Las sub-llaves se actualizan en cada vuelta. Todas las operaciones relacionadas con esta actualización de llave se pueden implementar a nivel de byte.

4.8.2.2 Procesadores de 32 bits

Los diferentes pasos que componen la transformación de cada vuelta se puede combinar mediante tablas, lo que permite unas implementaciones muy rápidas en procesadores con longitudes de palabra de 32 o más bits.

4.8.2.3 Paralelismo

Las cuatro transformaciones que se efectúan en cada una de las vueltas actúan en paralelo sobre los bytes, las columnas, las filas o un estado concreto. En la implementación con tablas, la mayoría de las operaciones XOR se pueden llevar a cabo en paralelo.

En aplicaciones donde el factor tiempo sea crítico, la llave expandida se calcula una única vez para un gran número de cifrados.

En aplicaciones donde la llave de cifrado cambie con bastante frecuencia, la obtención de la llave expandida y la ejecución de las vueltas del algoritmo se pueden realizar en paralelo.

4.8.2.4 Adaptación de hardware

El algoritmo Rijndael puede ser implementado en hardware diseñado específicamente para tal fin. Aún así, la implementación del algoritmo en software usando procesadores comerciales es muy rápida y el espacio ocupado en disco es pequeño, por lo que su implementación en hardware se reduciría a las siguientes características:

- Procesadores extremadamente rápidos sin restricciones de espacio. En este caso las tablas se pueden implementar en hardware y las operaciones XOR se pueden realizar en paralelo.

- Co-procesadores compactos en las tarjetas inteligentes para acelerar su ejecución.

4.8.3 Proceso de descifrado

En la implementación mediante tablas de valores es esencial que el único paso no lineal (operación *SubByte*) sea la primera transformación en cada vuelta y que los bytes de las filas sean permutados (*ShiftRow*) antes de aplicar la función *MixColumn*.

En la inversa de una vuelta concreta, el orden de las transformaciones es el contrario al orden de las operaciones establecido, por tanto, el paso no lineal será el último del proceso y los bytes de las filas serán permutados después de aplicar la inversa de la función *MixColumn*.

La inversa de una vuelta no se puede implementar con las tablas de valores del proceso de cifrado. Esto es una característica propia del diseño del algoritmo Rijndael. La estructura interna del algoritmo es tal que la secuencia de transformaciones de la inversa es igual al proceso de cifrado en sí, con las transformaciones reemplazadas por sus propias inversas y un cambio en la generación de llaves.

El orden de las operaciones *ShiftRow* y *ByteSub* es diferente. Esto es debido a que la operación *ShiftRow* afecta a la posición de los bytes, pero no modifica su valor. En el caso de *ByteSub*, el campo de trabajo es el valor de los bytes, independientemente de cuál se su posición en la matriz estado.

4.8.4 Implementación del descifrador Rijndael

La implementación del descifrador Rijndael es similar a la implementación del cifrador, ya que su estructura es también similar. Sin embargo, se aprecia una degradación en la ejecución del descifrador en procesadores de 8 bits. Esto se debe a que la ejecución del descifrador se considera menos importante. En muchos cifradores de bloque la operación de descifrado ni siquiera se implementa.

En procesadores de 32 bits la eficiencia del proceso de descifrado es la misma y no se produce ningún retardo con respecto al de cifrado.

4.8.4.1 Adaptabilidad al hardware

El cifrador y el descifrador del Rijndael se tienen que implementar en distinto hardware debido al empleo de las transformaciones en distinto orden. Se pueden compartir algunos circuitos concretos para algunas operaciones comunes a ambos procesos, pero no se puede implementar la totalidad de los dos algoritmos compartiendo todo el hardware.

4.8.5 Incremento del número de vueltas

El número de vueltas se ha determinado en función del máximo número de vueltas para el cual se pueden prevenir los *ataques shortcut*[73] con un considerable margen de seguridad. Un ataque *shortcut* es más eficiente que uno por fuerza bruta.

Para ejecuciones del algoritmo Rijndael empleando una longitud de bloque y de llave de 128 bits, no ha sido encontrado ningún *ataque shortcut* para implementaciones con más de 6 vueltas. Para conseguir un margen de seguridad se han añadido cuatro vueltas más, por lo que para llaves y bloques de 128 bits se emplean 10 vueltas. Los motivos de esta decisión han sido los siguientes:

- Dos vueltas del algoritmo Rijndael proporcionan una “alta difusión” en el sentido de que cada bit de un estado depende de los valores de todos los bits de las dos vueltas anteriores, o lo que es lo mismo, un cambio en el valor de un bit de un estado concreto afecta a todos los bits de los estados en las dos siguientes vueltas. La alta difusión de cada

[73] Ver Anexo I

una de las vueltas de Rijndael depende de la estructura uniforme que opera en todos los bits de un estado.

- Generalmente los criptoanálisis lineales, los criptoanálisis diferenciales y los ataques con truncated differencials (truncales diferenciales) provocan una propagación a través de n vueltas con el fin de atacar n + 1 ó n + 2 vueltas. La adición de cuatro vueltas adicionales dobla el número de vueltas a través del cual se puede dar una propagación.

Para versiones del algoritmo con llaves de mayor longitud, el número de vueltas aumenta en una unidad por cada 32 bits adicionales en la llave de cifrado principal. Esto se debe a las siguientes razones:

- Uno de los objetivos principales es la erradicación de los ataques shortcut, es decir, ataques que son más eficientes que una búsqueda exhaustiva de la llave de cifrado. Cuanto mayor sea la llave, un ataque shortcut tendrá más carga de trabajo, por lo que se vuelve más ineficiente.

- Los ataques empleando el conocimiento de parte de la llave principal o los ataques empleando llaves relacionadas se basan en el conocimiento de los bits de la llave principal de cifrado o en la habilidad para probar reiteradamente distintas llaves de cifrado. Si la llave de cifrado aumenta de tamaño, el rango de posibles llaves también aumenta, dificultando así el trabajo del criptoanalista.

Se ha encontrado que dichas extensiones de los ataques en una simple vuelta están muy lejos de provocar cualquier fallo en la seguridad.

4.8.6 Llaves débiles

Las llamadas llaves débiles son aquellas que resultan de un cifrador de bloque diseñado con debilidades detectables. El caso más conocido de llaves débiles se da en el algoritmo IDEA. Estas debilidades son características de cifradores en los cuales las operaciones no–lineales dependen del valor actual de la llave. Este no es el caso de Rijndael, donde las llaves se aplican usando funciones XOR y todos los elementos no-lineales se encuentran agrupados en las cajas de sustitución. En Rijndael no existen restricciones a la hora de seleccionar una llave.

4.8.7 Fortaleza estimada

El ataque más eficiente contra la seguridad de Rijndael es un ataque por fuerza bruta. Emplear pares de texto claro–texto cifrado conocidos para obtener información sobre otros pares no es más eficiente que el proceso de determinar la llave mediante una búsqueda exhaustiva, para lo cual el trabajo necesario depende de la longitud de la llave principal de cifrado de tal forma que:

Longitud de la llave (bytes)	Aplicaciones de Rijndael necesarias
16	2^{127}
24	2^{191}
32	2^{255}

Tabla 4.17 Fortaleza estimada de Rijndael

4.8.8 Metas de seguridad

Un criptoanalista se considera efectivo si es capaz de demostrar que una de las metas de seguridad descritas a continuación no es efectiva:

4.8.8.1 Seguridad K

Definición: Un cifrador de bloques es k-seguro si todas las posibles estrategias contra él tienen el mismo factor esperado de trabajo y requisitos de almacenamiento, como para la mayoría de los posibles cifradores de bloque con las mismas dimensiones.

La seguridad K es una noción de seguridad muy fiable. Se puede apreciar fácilmente que si una de las siguientes debilidades es apreciable en un cifrador, este no es k-seguro:

Existencia de ataques de descifrado de llave más rápidos que un ataque por fuerza bruta.

- Ciertas propiedades de simetría en el desarrollo
- Existencia de llaves débiles (como en IDEA)
- Ataques mediante llaves relacionadas

La seguridad K es esencialmente una medida relativa. Es posible diseñar un cifrador de bloque con un tamaño de bloque y de llave de 5 bits. La carencia de seguridad ofrecida por un esquema es debido a sus pequeñas dimensiones, no al hecho de que el esquema falle. Claramente, el aumento de la longitud de la llave implica el aumento en los requisitos de seguridad.

4.8.8.2 Cifradores de bloque herméticos

Es posible imaginar cifradores que tengan ciertas debilidades y sigan siendo k-seguros.

Un cifrador de bloque es hermético si no presenta debilidades que no se hallen presentes en la mayoría de los cifradores de bloque con las mismas longitudes de bloque y de llave.

Es decir que un cifrador es hermético si su estructura interna no puede ser explotada en ninguna operación.

4.8.8.3 Metas

Para cualquier longitud de bloque o de llave, las metas de seguridad de Rijndael son:

- Seguridad K
- Hermetismo

Si Rijndael cumple sus objetivos, la fortaleza contra cualquier ataque, ya sea conocido o desconocido, es tan buena como la de cualquier otro cifrador de bloque con las dimensiones dadas.

4.8.9 Ventajas y limitaciones de Rijndael

Ventajas

Rijndael se puede implementar para ser ejecutado a velocidades inusualmente elevadas para algoritmos de cifrado de bloque.

Rijndael se puede implementar en tarjetas inteligentes, empleando un mínimo de memoria RAM y usando un número reducido de vueltas.

Las operaciones de cada vuelta se pueden paralelizar, lo cual supone un importante avance con vistas a futuros procesadores y hardware específico.

Como el algoritmo no hace uso de operaciones aritméticas, no existe ningún prejuicio con respecto a ningún tipo de arquitectura de procesadores.

Simplicidad del diseño

El algoritmo es autónomo o independiente. No hace uso de ningún otro componente criptográfico, cajas de sustitución procedentes de otros algoritmos o tablas aleatorias.

El algoritmo no basa su seguridad o parte de ella en operaciones aritméticas ininteligibles o secretas.

El diseño hermético del algoritmo no permite la existencia de puertas traseras ocultas (backdoors).

Longitud variable del bloque

Las longitudes de bloque de 192 y 256 bits permiten la construcción de una función resumen usando Rijndael como función de compresión y con la característica de que el número de colisiones provocadas sería mínimo. Hoy en día, la longitud de bloque de 128 bits no se considera suficiente para dicho fin.

Extensiones

El diseño permite la especificación de variaciones con la longitud del bloque y de la llave en el rango comprendido entre los 128 y 256 bits, tomados en intervalos de 32 bits.

Aunque el número de vueltas del Rijndael esta fijado en las especificaciones del algoritmo, se puede modificar este parámetro con el fin de solucionar posibles problemas de seguridad o adaptarse a distintas velocidades de ejecución.

Limitaciones

Las limitaciones del algoritmo están relacionadas con su inversa:

El descifrador Rijndael es más difícil de implementar en tarjetas inteligentes que el propio algoritmo debido a que requiere una mayor cantidad de código y un número superior de vueltas. A pesar de ello, comparado con otros cifradores, la función inversa es realmente rápida.

En implementaciones de software, el algoritmo y su inversa emplean diferente código y diferentes tablas.

En implementaciones de hardware, el algoritmo solamente puede aprovechar una parte de la circuitería implementada para el algoritmo.

4.9 Comparación entre el DES y el Rijndael

El algoritmo DES es el resultado de una modificación del algoritmo *Lucifer*, presentado al concurso convocado por el NBS en 1974.

El algoritmo Lucifer contaba con una llave de 128 bits, un valor realmente seguro para la fecha que, por motivos que jamás han sido publicados oficialmente, el NBS decidió que el nuevo estándar sería una versión del Lucifer con una llave reducida a sólo 64 bits. Se dice que el motivo de tal reducción fue que, con las computadoras más potentes de aquellos años, solamente se podía llegar a romper el secreto de un criptograma con una llave de, como mucho, 64 bits. Además, observando el desarrollo del algoritmo, se puede aprecia que, de esos 64 bits, 8 de ellos son de paridad, y que, tras pasar por las cajas de sustitución, el algoritmo trabaja internamente con una llave de únicamente 48 bits.

Tras veinte años de vida, el tamaño de la llave de DES, que durante todo este tiempo ha garantizado el secreto de la información, ha sido relegado a un segundo plano por el avance imparable de la potencia de las computadoras. Los 64 bits de longitud de la llave se han quedado obsoletos frente a los nuevos algoritmos de cifrado, que emplean llaves de mayor tamaño.

Se debe tener en cuenta que el DES lleva ya más de veinte años en servicio, y que la escasa longitud de su llave, comparándola con la de otros cifradores actuales, se debe al tiempo transcurrido desde que se diseñó y no a un diseño deficiente.

4.9.1 Ámbitos de uso

Ambos algoritmos fueron diseñados con el fin de proporcionar una plataforma de cifrado potente y útil en multitud de tareas distintas, que abarcan desde el cifrado de mensajes en una red de comunicaciones hasta la codificación de señales vía satélite. El abanico de aplicaciones posibles es de tal tamaño que se podrían incluir en cualquier tipo de proceso.

Aunque obviamente su función principal es cifrar información, preservando el secreto de la misma, actualmente estos algoritmos se utilizan con otros fines, muy distantes del cifrado pero para los cuales se ha visto que son algoritmos apropiados. Uno de estos fines es el uso como función resumen. En este contexto, se hace uso de la propiedad de los algoritmos de cifrado que nos dice que el uso de llaves de cifrado similares no implica cifrados similares, es decir, la posibilidad de que al aplicar distintas llaves a distintos datos proporcione un cifrado idéntico es ínfima.

Otro uso bastante corriente de estos algoritmos es el uso de los mismos como generadores de números pseudoaleatorios.

Gracias a la facilidad que aportan ambos algoritmos para su implementación tanto en hardware como en software y gracias a la multitud de entornos sobre los cuales se pueden ejecutar, gran variedad y tipo de procesadores, tarjetas inteligentes, sistemas operativos, etc., estos dos

algoritmos han pasado a formar parte de nuestra vida cotidiana y forman un eslabón esencial de la cadena de información en la que se ha convertido la sociedad actual. Por ejemplo, estos algoritmos se emplean en la codificación de los canales de televisión por cable, en las llamadas telefónicas, en las sucursales bancarias e incluso en las tarjetas de crédito.
Ambos algoritmos pueden operar en cuatro modos diferentes. Estos modos fueron definidos para el algoritmo DES por el propio NIST, mientras que en el caso de Rijndael, aunque la normativa del concurso convocado por el NIST no especificaba concretamente estos cuatro modos de uso, dejaba claro que el ganador pasaría a ser el nuevo estándar de cifrado, por lo que debería suplir al DES en todos sus ámbitos de uso.

Estos cuatro modos de operación son los siguientes:

- Electronic CodeBook (ECB)
- Cipher Block Chaining (CBC)
- Cipher FeedBack (CFB)
- OutPut FeedBack (OFB)

El modo ECB hace referencia al empleo del algoritmo directamente como método de cifrado y descifrado de datos.

El modo CBC es una variante del algoritmo que trabaja con varios bloques de datos simultáneamente.

El modo CFB emplea un texto cifrado anteriormente como entrada del algoritmo con el fin de generar una salida pseudoaleatoria que a su vez se combina con el texto en claro para obtener el criptograma, consiguiendo así una modificación del resultado final.

El modo OFB se comporta de manera idéntica al modo CFB, a excepción de que en este modo se emplea la propia salida del algoritmo como entrada del mismo, mientras que en el caso anterior se empleaba un texto cifrado previamente.

Ambos algoritmos poseen una gran adaptabilidad a cualquier software y hardware, sin embargo, se puede observar que el descifrador del algoritmo DES es más fácil de implementar y computacionalmente más eficiente que el descifrador de Rijndael. La diferencia radica en que para descifrar un texto cifrado por el DES únicamente se aplican las sub-llaves de la red de Feistel al revés, es decir, en la vuelta 1 se empleará la sub-llave 16, en la vuelta 2 se empleará la sub-llave 15 y así sucesivamente, de forma que, no se necesita variar la estructura interna del algoritmo, lo cual también favorece su implementación en hardware.

En cuanto a Rijndael, el descifrador necesita un mayor número de ciclos de procesamiento que el cifrador, por lo que es menos eficiente. Además, aunque la estructura interna de cada vuelta no ha variado en cuanto a contenido, sí lo ha hecho en cuanto al orden de las funciones que la componen, por lo que esto perjudica la implementación en hardware, impidiendo la reutilización total del hardware en el que se ha implementado.

4.9.2 Llaves de cifrado

Rijndael cifra con tres tamaños de llave distintas cuyos valores son 128, 256 y 512 bits frente a la llave fija de 64 bits empleada por el DES. Además, esta llave de cifrado de 64 bits, no es usada en su totalidad. De los 64 bits de la llave, ocho de ellos son los llamados bits de paridad, que no intervienen en el proceso de cifrado sino que fueron introducidos con los siguientes fines:

- Detección de errores
- Obtener un mismo valor de tamaño tanto para la llave como para el bloque de cifrado

La posibilidad ofrecida por Rijndael de variar el tamaño de la llave permite al usuario del sistema, ya sea un usuario final propiamente dicho o un administrador de un sistema informático, modificar las propiedades del algoritmo consiguiendo una adaptación óptima a sus necesidades, lo que lo hace ideal para multitud de funciones.

En la ejecución de cualquier algoritmo de cifrado, se identifican dos parámetros, directamente apreciables por el usuario, pero a su vez mutuamente excluyentes:

- Velocidad
- Fortaleza

Las dos cualidades son exigibles a un buen algoritmo de cifrado, aunque por desgracia el aumento de una de ellas implica la disminución de la otra. Cuanto mayor sea el tamaño de la llave principal de cifrado, más difícil será de romper el secreto que el cifrado guarda, aunque, esto implica que el proceso de cifrado de la información precisará de un mayor número de ciclos de procesamiento, ya que el número de bits y el número de las operaciones que se deben procesar será superior. El usuario del algoritmo debe elegir cual de los dos parámetros es prioritario para el uso que está haciendo del algoritmo. Se debe emplear el máximo tamaño posible de llave en procesos en los cuales el tiempo no es un factor a tener en cuenta y se da prioridad a la seguridad, estos suelen ser procesos donde se ejecuta pocas veces el algoritmo y casi siempre empleando la misma llave de cifrado. El reto al que se enfrenta un criptoanalista a la hora de realizar un ataque contra este tipo de procesos es que la longitud de la llave impide ataques por fuerza bruta o, sea cual sea la estrategia de ataque empleada, necesitará muchos ciclos de procesamiento, o lo que es lo mismo, tiempo.

Frente a este tipo de procesos lentos, existen también procesos en los que un tiempo de cifrado reducido es indispensable. Suelen ser procesos en los que se cifran informaciones de pequeño tamaño, pero que se están produciendo constantemente y cada uno con una llave de cifrado distinta. Un ejemplo de esto sería el proceso de cifrado que sufren los paquetes de datos dentro de una red de computadoras entre el servidor y los distintos clientes. En este tipo de procesos, el tamaño de los mensajes es mínimo y la cantidad de mensajes a cifrar es elevada, por lo que el proceso de cifrado debe ser lo más rápido posible para evitar introducir retardos en la red, por tanto una llave relativamente pequeña es suficiente para garantizar la seguridad, ya que un criptoanalista se encontrará con que una vez que haya conseguido descifrar la llave del mensaje (si lo consigue, ya que es bastante improbable y además la dificultad que supone conseguir ese mensaje cifrado de la red y el tiempo empleado en ello), el receptor ya habrá recibido el mensaje. Además, cada uno de estos mensajes se cifra con una llave distinta, por lo que para descifrar el siguiente mensaje, se debe volver a empezar el proceso desde el principio, ya que todo el trabajo realizado anteriormente para descifrar la llave, no será reutilizable. En este sentido, Rijndael permite adaptar la velocidad de ejecución del algoritmo a necesidades específicas, mejorando la velocidad mediante el uso de una llave de 128 bits, seleccionando una fortaleza extrema mediante la llave de 256 bits o empleando un valor intermedio de 192 bits.

Cuando se diseñó el DES se buscaba un tamaño de llave general que permitiese que el algoritmo se pudiera emplear para multitud de funciones, pero en este sentido no se tuvieron en cuenta las prestaciones específicas para los usos concretos. Se diseñó para conseguir unas buenas prestaciones y una buena relación velocidad–seguridad bajo cualquier tipo de trabajo, pero no se

tuvo en cuenta el uso para casos extremos, donde la posibilidad de variar el tamaño de la llave puede introducir importantes mejoras en la calidad final del cifrado o en el funcionamiento, por ejemplo, en una red de computadoras con un elevado transito de mensajes.

Los creadores de Rijndael ofrecen un punto de vista en el que sustituyen el uso de una sub-llave específica de cada vuelta, por la aplicación de 4 sub-llaves de Nb bytes en cada vuelta, una para cada fila de la matriz de datos. En la siguiente vuelta se seleccionan las siguientes 4 sub-llaves. Así, en cada vuelta se rellena una matriz que será la sub-llave de esa vuelta, que tendrá igual tamaño que el bloque de datos. Esto es equivalente a seleccionar una única llave de 4 * Nb bytes.

En cuanto a la generación de sub-llaves, se encuentran similitudes entre Rijndael y DES, ya que ambos algoritmos se engloban dentro de la categoría de cifradores producto. Esto es, generan las sub-llaves a partir de una única llave principal mediante una serie de funciones concretas.

Para suplir el pequeño tamaño de llave del DES y su debilidad frente a la potencia computacional demostrada por las máquinas actuales, se creó un nuevo estándar de cifrado llamado Triple DES, basado en la aplicación reiterada del DES, consiguiendo así una llave de cifrado de 168 bits. Este algoritmo soluciona el problema del tamaño de la llave de su predecesor, pero debido a que se basa simplemente en aplicar reiteradamente el DES y no en un nuevo diseño, hereda todos los problemas de dicho algoritmo acerca de la llave y su adaptabilidad al ámbito de uso, mencionados anteriormente. Además, el valor de llave de TDES garantiza la confidencialidad de la información frente a la potencia de las computadoras actuales, aunque esa llave de 168 bits se quedará obsoleta en un breve periodo de tiempo.

4.9.3 Tamaño de bloque de datos

En cuanto al tamaño de bloque empleado en el proceso de cifrado o descifrado el DES utiliza un bloque de datos con un tamaño de 64 bits, frente a Rijndael, que se puede ejecutar sobre tres tamaños de bloque diferentes; 128, 192 y 256 bits. Es decir que Rijndael trabaja con bloques de mayor tamaño que DES y por tanto se incrementa la dispersión de la información, ya que cuanto mayor sea el tamaño de bloque, mayor será la dispersión y la dificultad para romper el secreto.

El hecho de que Rijndael pueda trabajar con tres tamaños distintos de bloque, aporta las siguientes ventajas:

4.9.4 Aumento en la seguridad

Un criptoanalista tendrá que enfrentarse, además del problema de que el mayor tamaño de bloque aumenta la difusión de la información, al problema que le supone el hecho de hallar cuál de los tres tamaños de bloque posible ha sido empleado en el proceso de cifrado. Rijndael puede combinar cualquiera de sus tres posibles tamaños de llave con cualquiera de los tres tamaños de bloque, es decir que existen nueve posibilidades distintas de combinación llave–bloque. Esto provoca que un ataque tenga que, para hallar la llave, trabajar con los tres tamaños posibles de bloque, por lo que el trabajo que esto conlleva provoca un incremento en el tiempo empleado para romper el secreto de la información, por lo que, al aumentar el tiempo necesario para descifrarlo, se aumenta la propia seguridad de la información. En DES, al poseer únicamente un tamaño de bloque posible, se elimina esta variable de la ecuación, por tanto el tiempo necesario para romper el secreto será menor, ya que solamente posee una combinación llave–bloque posible y para descubrir la llave secreta no se tendrá que tomar el factor bloque en cuenta, ya que es una constante de 64 bits.

4.9.4.1 Variación de la velocidad

El usuario del cifrador Rijndael puede seleccionar cualquiera de los tres tamaños de bloque (128, 192 y 256 bits) para cifrar sus datos. Cuanto mayor sea este tamaño, mayor será la difusión de la información. Como resultado, se incrementa la seguridad y la velocidad de ejecución del algoritmo ya que la cantidad de información procesada de una sola vez es mayor, por lo que se pierde menos tiempo en procesos de entrada–salida de datos. La posibilidad de tener tres tamaños de bloque de datos diferentes permite al usuario una mejor adaptación del proceso de cifrado al medio en el que se ejecute el algoritmo, seleccionando un tamaño de bloque pequeño para el caso en el que la capacidad computacional sea menor, y un tamaño de bloque grande para casos en los que la velocidad de ejecución sea un factor relevante.

El hecho de seleccionar el menor tamaño de bloque posible en Rijndael (128 bits) no implica, en ningún caso, que el algoritmo deje de ser seguro y comprometa la información cifrada. Se ha demostrado que el algoritmo Rijndael con un tamaño de bloque de 128 bits e incluso combinado con un tamaño de llave de 128 bits, sigue siendo totalmente seguro y con un amplio margen de seguridad.

4.9.5 Estructura interna de los algoritmos

4.9.5.1 Representación de la información

El algoritmo DES representa la información mediante la numeración ordenada de los bits que la componen, ya sea la información de la llave o del bloque de datos. Suele usar los números naturales empleando entre ellos las operaciones de suma, multiplicación y la operación lógica XOR.

En el algoritmo Rijndael la unidad básica de información es el byte o el conjunto de bytes, aunque, obviamente, también se trabaja a nivel de bit. Para la representación de bytes o de bits se ha elegido la representación mediante polinomios con coeficientes pertenecientes a $GF(2^8)$ debido a las ventajas que el uso de polinomios supone.

4.9.5.2 Permutaciones empleadas

Una herramienta esencial en el cifrado es la permutación. Ambos algoritmos, al igual que la mayoría de los algoritmos de cifrado existentes, hacen uso de este recurso, pero de distinta forma.

El algoritmo DES aplica unas permutaciones tanto a la llave de cifrado como al bloque de datos, efectuando esta operación a nivel de bit. Estas permutaciones vienen dadas por unas tablas, por lo que la permutación será la misma sea cual sea el valor de los bits que componen la llave o los que componen el bloque de datos.

El algoritmo Rijndael no aplica permutaciones a ninguna llave, ya sea la llave principal o cualquiera de las sub-llaves empleadas. Únicamente aplica permutaciones al bloque de datos, proceso que se lleva a cabo en la función ShiftRow. Además del menor número de permutaciones empleado comparando Rijndael con DES, es destacable que la permutación efectuada en ShiftRow es una permutación a nivel de byte, mientras que las permutaciones en el algoritmo DES se efectúan a nivel de bit. Otra diferencia notable es que el valor de la permutación no es fijo como en DES sino que depende directamente del valor de N_b, es decir, depende directamente del tamaño de bloque seleccionado por el usuario. Esta posibilidad no puede darse en el algoritmo DES, ya que el tamaño de bloque es fijo, no variable como en Rijndael.

4.9.5.3 Estructura de la red

El algoritmo DES adopta una red Feistel. Una vez que se ha aplicado la permutación inicial al bloque de datos, se inicia un proceso de cifrado con sub-llaves de una forma reiterativa. En cada una de estas vueltas, el bloque de texto se divide a la mitad. Sobre la mitad derecha se aplica una serie de operaciones mientras que la mitad izquierda no sufre ninguna variación. Posteriormente, al final de cada vuelta, se permutan las mitades derecha e izquierda para así, en la siguiente vuelta, poder trabajar con la mitad del bloque que en esta vuelta ha quedado sin ninguna modificación.

Rijndael adopta una estructura similar, empleando un proceso repetitivo a lo largo de una serie de vueltas, formando una red, pero esta red no se puede clasificar como una red tipo Feistel. En este algoritmo la totalidad de los bits que componen el bloque de texto son susceptibles de ser empleados por las operaciones del algoritmo, es decir, las operaciones de la vuelta afectan a la totalidad de los bits del bloque de datos. Los bits que componen el bloque de datos, en una situación de igualdad de vueltas, reciben el doble de tratamiento en una red del tipo de la empleada en el algoritmo Rijndael que en una red tipo Feistel.

En el caso del algoritmo DES, el número de vueltas de la red Feistel es igual a 16, por lo que cada bit se trata, únicamente, 8 veces, mientras que en Rijndael un bit es tratado tantas veces como vueltas tenga la red.

El número de vueltas de la red Feistel del algoritmo DES es inalterable, y no se puede variar el número de vueltas sin modificar la estructura interna del algoritmo, por lo que una modificación de este tipo afectaría su diseño y podría comprometer la fortaleza del mismo. El impedimento a la modificación del número de vueltas de la red Feistel viene dado por el método que emplea el DES para la generación de sub-llaves. En este algoritmo, una vez quitados los bits de paridad y reducida la llave principal a 48 bits, ésta se divide en dos mitades. Para calcular una sub-llave, se toman las dos mitades de la sub-llave anterior y se rotan uno o dos bits a la izquierda según unos valores dados.

En Rijndael el número de vueltas del algoritmo (N_r) depende de los valores de N_k y N_b, que a su vez dependen, respectivamente, del tamaño de la llave y del tamaño de bloque seleccionado por el usuario. Cuanto mayor sea el tamaño de bloque y el tamaño de llave, mayor ha de ser el número de vueltas del algoritmo. Los diseñadores de este método han seleccionado valores comprendidos entre 10 y 14 para el número de vueltas.

Una de las principales críticas al diseño de Rijndael fue el relativamente reducido número de vueltas de su red interna, aunque este parámetro se puede variar sin dificultad. Se podría emplear un número mayor de vueltas, para casos de extrema necesidad de privacidad.

Como se puede apreciar, Rijndael vuelve a ofrecer un entorno totalmente configurable y adaptable al ámbito de uso. Los diseñadores se limitan a sugerir unos valores adecuados, aunque el diseño del algoritmo permite una modificación de estos valores, de manera inmediata, simplemente modificando el valor de una constante. Esto supondría que, una vez tomada la decisión de modificar este parámetro, el sistema podría volver a estar operativo en cuestión de minutos, el tiempo que lleve volver a compilar el algoritmo, por lo que la interrupción del sistema sería mínima.

El diseño del Rijndael permite la posibilidad de su paralelización, aprovechando así las posibilidades de cálculo de máquinas que posean varios procesadores y mejorando así el tiempo de cifrado y descifrado del algoritmo y facilitando las implementaciones en hardware específico.

Frente a esto se encuentra la linealidad de cada una de las vueltas de la red Feistel del algoritmo DES, cuyo diseño no introdujo componentes para una ejecución paralela, aunque se debe mencionar también que las vueltas de la red Feistel son mucho menos complejas que las vueltas de la red propuesta por el algoritmo Rijndael, ya que el número de operaciones de las vueltas de Rijndael es mayor, al igual que el número de bits a los que estas operaciones afectan.

4.10 Resistencia frente a los principales ataques

4.10.1.1 Criptoanálisis diferencial

El criptoanálisis diferencial se ve favorecido por la estructura de la red Feistel del DES, ya que la mitad izquierda del bloque de texto a la entrada de una vuelta determinada coincide con el bloque derecho a la entrada de la vuelta anterior por lo que los patrones de diferencias sólo afectan a parte de las ocho cajas de sustitución del algoritmo, por lo que es susceptible de que un ataque mediante este método tenga éxito en un tiempo razonable. El diseño del algoritmo DES no estaba preparado para soportar este tipo de ataques que fueron desarrollados por Eli Biham y Adi Shamir en 1991, casi quince años después del desarrollo de DES.

Eli Biham y Adi Shamir demostraron que dado un sistema de cifrado DES con la llave asignada, se podía llegar a deducir la llave principal de cifrado mediante unas reducciones estadísticas basadas en los resultados de hacer pasar un total de 247 mensajes concretos a través de las cajas de sustitución del algoritmo. A pesar de que este método de ataque es capaz de comprometer la seguridad del algoritmo DES, no logra romperlo con la facilidad que se supone debería un algoritmo diseñado con anterioridad a la creación del ataque, es más, muchos de los algoritmos modernos de cifrado son menos resistentes que el DES ante dicho ataque. Este hecho, junto a que los motivos que llevaron a la elección de las cajas de sustitución del DES han sido declarados como secreto nacional por las autoridades norteamericanas, ha llevado a creer que durante el desarrollo de DES se había previsto un ataque de este tipo, aunque no se había llegado a concretar.

Es importante recalcar la ventaja de la juventud del algoritmo Rijndael, por lo que sus diseñadores ya han previsto la fortaleza del algoritmo frente a este ataque.

4.10.1.2 Criptoanálisis lineal

En general, un ataque por criptoanálisis lineal contra el algoritmo DES o algún otro método de cifrado basado en él, como puede ser el TDES, se aprovecharía de que la mitad izquierda de los bits de salida de una vuelta son los mismos que la mitad derecha de los bits de entrada en esa vuelta.

En el caso del algoritmo Rijndael, se ha probado que este tipo de ataque no es efectivo siempre y cuando no se implemente una versión del algoritmo con menos de cuatro vueltas en la red interna.

4.10.1.3 Ataques por análisis temporal (*Timing attacks*)

Este ataque fue diseñado y desarrollado por Paul Kocher. En Rijndael, el uso de polinomios con coeficientes pertenecientes a GF(2^8) favorece la fortaleza del algoritmo frente a ataques por análisis temporal.

El éxito de un ataque por análisis temporal reside en las multiplicaciones matemáticas. En Rijndael sólo se lleva a cabo una multiplicación matemática en cada una de las vueltas, ya que el resto son operaciones lógicas. Esta operación se lleva a cabo en la función MixColumn. Este podría ser el único punto débil frente a ataques por análisis temporal, evitándolo por medio de coeficientes pertenecientes a GF(2^8).

El algoritmo DES también es resistente frente a los ataques por análisis temporal aunque, la diferencia de Rijndael, su resistencia no se basa en el modelo de representación de coeficientes. El DES no utiliza multiplicaciones matemáticas. Los dos algoritmos proporcionan una plataforma de cifrado robusta frente a ataques por análisis temporal, aunque, mientras Rijndael basa su fortaleza en la calidad del diseño interno, DES basa la suya en las limitaciones a la hora de implementar funciones matemáticas.

4.10.1.4 Ataque mediante la existencia de llaves débiles

En la inmensa mayoría de los casos, los conjuntos M (Mensaje de texto claro) y C (Mensaje cifrado) deben ser iguales. Esto quiere decir que tanto los "textos en claro" como los "textos cifrados" se representan sobre el mismo alfabeto.

Se puede dar el caso de que ciertas llaves concretas generen textos cifrados de poca calidad. Una posibilidad bastante común en ciertos algoritmos es que algunas llaves tengan una propiedad que hace que el texto cifrado recupere el "texto en claro" original. Estas circunstancias podrían llegar a simplificar enormemente un intento de vulnerar el sistema, por lo que habría que evitarlas a toda costa.

La existencia de llaves con estas características, depende en gran medida de las peculiaridades de cada algoritmo en concreto, y en muchos casos también de los parámetros escogidos a la hora de aplicarlo. Las llaves que no cifran correctamente los mensajes son llaves débiles. Normalmente en un buen algoritmo de cifrado la cantidad de llaves débiles es nula o muy pequeña en comparación con el número total de llaves posibles.

El algoritmo DES presenta algunas llaves débiles y llaves semi-débiles. En cualquier caso, el número de llaves de este tipo es tan pequeño en comparación con el número total de posibles llaves, que no debe suponer un motivo de preocupación.

4.11 De la criptografía simétrica a la asimétrica

En cualquier método de cifrado simétrico o de llaves privadas, los comunicantes necesitan en algún momento intercambiar de forma segura una llave secreta. Al ser dicha llave un secreto compartido, existe la posibilidad de que alguno de los comunicantes suplante a otro si no se toman medidas de seguridad adicionales. Asimismo, puesto que el intercambio de llaves debe hacerse de forma segura, éste requiere la utilización de correos especiales o incluso valijas diplomáticas.

En consecuencia, cabe hacerse la pregunta de cómo garantizar la seguridad en la transmisión o intercambio de las claves secretas. Para resolver este problema se inventó la denominada criptografía asimétrica o de llave pública descrita en detalle en el capítulo siguiente. Con todo ello, la criptografía simétrica es la más utilizada a nivel mundial para garantizar la confidencialidad de los mensajes transmitidos, debido a su seguridad y alta velocidad de cifrado. Sin embargo, este tipo de criptografía raramente se utiliza para llevar a cabo la transmisión de las llaves secretas entre los comunicantes, para lo cual se utiliza la criptografía de llave asimétrica antes referida.

5

Algoritmos de llave pública

Tal como se mencionó al final del capítulo 2, los algoritmos de llave pública rompieron en los años 70 con un paradigma en el ámbito de la criptografía. En este capítulo se profundiza en el estudio de ese tipo de algoritmos.

5.1 El protocolo Diffie Hellman para intercambio de llaves

Si se tienen 10 amigos con los que se desea establecer comunicación, se puede convocarlos a una reunión donde todos puedan intercambiar sus llaves secretas, pero, como todas las llaves, éstas serán actualizadas periódicamente, entonces, todas las personas se tendrían que reunir nuevamente cada vez que deseen intercambiar sus llaves privadas. Pero, según el grupo de amistades crezca, el número de llaves también crecerá, de manera que la situación se volverá inmanejable.

Los matemáticos Whitfield Diffie y Martin Hellman plantearon la pregunta, de sí sería posible recolectar las llaves de una manera más eficiente, a lo que ellos mismos propusieron la creación de un algoritmo para el cual la llave de cifrado y descifrado sean diferentes. Se podría publicar la llave de cifrado y mantener la llave de descifrado en secreto, así, cualquier podría enviar mensajes cifrados y solo el dueño de la llave de descifrado podría descifrarlos. Esto resolvería el problema de la distribución de muchas llaves diferentes.

Aunque Diffie y Hellman hicieron el planteamiento, ellos sólo podían proveer una respuesta parcial, la cual actualmente es conocida como El protocolo Diffie Hellman para intercambio de llaves (*Diffie–Hellman Key Exchange Protocol*), algunas veces abreviado como el protocolo DH. Dicho protocolo indica:"Dos personas que se comunican a través de un canal inseguro pueden

acordar una llave secreta, de manera que ambos reciban la misma llave sin que alguien que interfiera el canal y escuche su conversación la pueda obtener".

Mientras Hellman estaba trabajando en el algoritmo de intercambio de llaves, Diffie continuaba su investigación. Su idea de intercambio de llaves era factible, sin embargo, tenía limitaciones prácticas: ambas partes debían primero crear su llave secreta en un proceso mutuo. Como consecuencia, el remitente tenía que esperar hasta que la llave común fuera creada para enviar el mensaje cifrado. Diffie trató de contrarrestar lo anterior, proponiendo lo siguiente: Cada parte debía procesar un par de llaves, una pública y una privada. Por ejemplo, la llave pública de María es usada por Pedro para cifrar el mensaje para María considerando que su llave privada será utilizada para descifrar el mensaje cifrado de Pedro.

Aunque no fue Diffie quien diseñó el algoritmo que llevara a la realidad su idea de llave pública o privada, él merece el crédito por hacer lo inconcebible concebible además inició un grupo de investigación con el fin de encontrar una función matemática que reflejara su visión. La pregunta para el equipo de investigadores era:

> *¿Qué función matemática permitiría a cualquier persona cifrar un mensaje secreto para un destinatario usando la llave pública del mismo (destinatario) y previniendo que nadie más que él pueda descifrar dicho mensaje?*

Requeriría de un matemático experimentado el encontrar la respuesta a la pregunta anterior, una función resumen era la solución, ya que la mayoría de funciones pueden ser invertidas. Por ejemplo: Los desplazamientos a la derecha del método de César pueden ser invertidos, realizando desplazamientos a la izquierda. Matemáticamente, una suma puede ser invertida por una resta, esta es la función ejemplo más simple.

Los ganadores del reto anterior, una función resumen de un solo sentido, fueron 3 israelíes, graduados de MIT, Ronald Rivest, Adi Shamir y Leonard Adleman, quienes presentaron sus resultados en 1978[74].

La ironía del proceso de creación del RSA (**R**ivest, **S**hamir, **A**dleman) es que, Rivest y Adleman originalmente trataron de probar que la idea de Diffie de una llave pública y una privada no podía ser llevada a cabo, ya que no existían funciones resumen apropiadas. Pero fallaron en su intento de demostrarlo, y accidentalmente tuvieron éxito: mientras Rivest revisaba las ideas de cifrado, Adleman trataba de atacarlas y Shamir les ayudaba[75]. Al final el RSA también incluía los esquemas de firmas digitales.

5.2 RSA

La creación de una llave pública común y una secreta fue el primer paso de la criptografía de llaves públicas. De manera que esto enunciaba que dos personas podían establecer comunicación sin tener que entregar la llave secreta con anterioridad.

Un sistema de cifrado de llaves públicas, es aquel, en el cual una llave, que se utiliza para cifrar mensajes, puede hacerse pública sin revelar la llave secreta que permite que los mensajes sean leídos.

---

[74] R.L. Rivest, A. Shamir, y L.M. Aldeman, *A Method for Obtaining Digital Signatures and PublicKkey Critposystemns*

[75] Diffie, W. *The First Ten Years of Public-Key-Cryptography*. Publicación de IEEE, Nueva York, 1992

Lo anterior confirma una de las propiedades de las llaves publicas: El conocimiento de la llave que cifra el mensaje no conduce al conocimiento de la llave que lo descifra.

Como consecuencia, sólo un par de llaves por persona es necesitado. Así que un total de n par de llaves es necesitado para n personas comunicándose. Es decir, 100 personas necesitarán 100 pares de llaves. Las 100 llaves que cifran los mensajes son públicamente conocidas y están listadas en directorios públicos en el Internet.

Actualmente, la mayoría de llaves utilizadas son generadas por el software "Pretty Good Privacy", PGP[76]. Las 100 llaves correspondientes al descifrado se deben mantener en un lugar absolutamente seguro, no en el disco duro de una computadora u otro dispositivo.

5.2.1 Seguridad del RSA

Lo que hace al RSA seguro son los números primos *p* y *q* que deben ser números de al menos 100 dígitos.

River, Shamir y Adleman sabían que la multiplicación de dos números grandes no es difícil, sin embargo, encontrar los factores de un entero grande es un problema muy difícil. Por ejemplo, es fácil calcular:

$$35 = 7 * 35$$
$$70 = 7 * 5 * 2$$
$$69 = 3 * 23$$
$$221 = 13 * 17$$
$$11413 = 113 * 101$$

Además, un programa de factorización rápidamente encuentra el número de 20 dígitos tal que:

$$10726291417797115873 = 1223233789 * 8768799157$$

Sin embargo, aún el mejor programa de factorización no podría encontrar el factor de 196 dígitos de un número como el siguiente:

1072629139784206448651876669948737985105534479415491719546499789238601953099099917195464891606235569002029020617284133518518580548148163084320989148925926075485185203509876558141111112930000042837

5.2.2 Firmas digitales y RSA

Además de eliminar el intercambio de llaves, probar la autenticidad de las personas era un reto que el RSA enfrentó a finales de los años 70.

Las firmas digitales del RSA utilizan las llaves de cifrado públicas y privadas que son también utilizadas para el proceso de cifrado y descifrado.

[76] Abordado al final del presente capítulo.

El cifrado con RSA requiere el uso de llaves de cifrado públicamente conocidas del destinatario del mensaje, sin embargo, la firma del RSA requiere que el emisor utilice su llave pública y privada.

Una razón adicional para la popularidad del RSA es la siguiente: No sólo puede cifrar o firmar digitalmente un documento, sino que, permite hacer ambas cosas. Primero lo cifra utilizando la llave pública del destinatario para posteriormente firmar el mensaje utilizando la llave privada del remitente.

5.2.3 Firmando un documento utilizando RSA

Un documento escrito a mano es firmando con un bolígrafo para probar la autenticidad del emisor, éste, utiliza una información personal única, un secreto que sólo él conoce a fin de probar su autenticidad. Similarmente, para firmar un documento digitalmente, el emisor deberá agregarle una información única que nadie más pueda tener o copiar. Esta información es su llave de descifrado o llave privada.

Si se parte de que no se puede obtener la llave privada partiendo de la llave pública, y que una firma hecha manualmente sobre un documento sí puede ser copiada o imitada, entonces se puede concluir que las firmas digitales son más confiables que las firmas manuales.

5.3 Usos del RSA

Las llaves largas vuelven lento el proceso de cifrado, de igual forma, con respecto a las firmas digitales se puede afirmar que: la longitud de la firma digital transmitida será igual a la longitud del mensaje. En otras palabras entre más largo sea el mensaje, más larga será la firma digital. Esta es una desventaja, si se compara con las firmas manuales, que son las mismas independientemente de la cantidad de información contenida en el documento.

El RSA es prácticamente usado para cifrar llaves de algoritmos de cifrado simétrico, este algoritmo es de uso práctico sí es utilizado junto a dicho cifrado.

5.4 Cifrado utilizando algoritmos de llave pública

A continuación, un ejemplo del proceso de utilización de algoritmos de cifrado de llave pública, los protagonistas del ejemplo y sus respectivas llaves se detallan en la siguiente tabla:

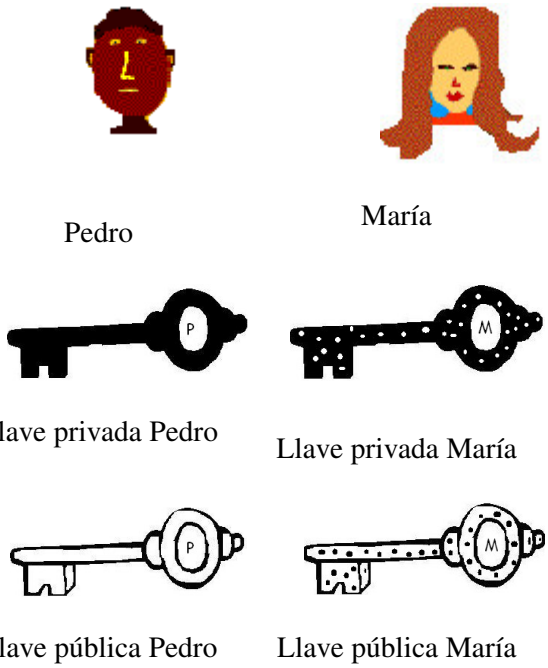


Tabla 5.1 Protagonistas del ejemplo con algoritmos de llave pública

María y Pedro tienen su respectivo par de llaves: una clave privada que sólo ha de conocer el propietario de la misma y una clave pública que está disponible para todos los usuarios.

1. María escribe un mensaje para Pedro y quiere que sólo él pueda leerlo. Por esta razón lo cifra con la llave pública de Pedro, la cual se encuentra disponible para cualquier persona.

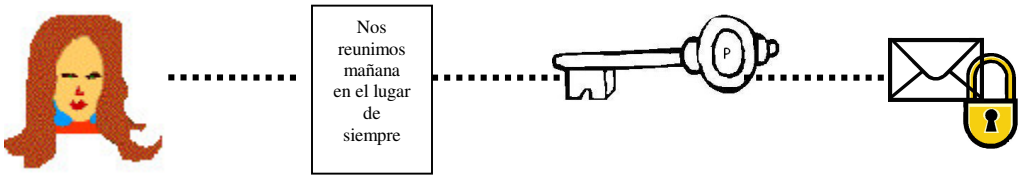


Figura 5.1 Mensaje cifrado de María para Pedro

2. Se produce el envío del mensaje cifrado no siendo necesario el envío de la clave.

3. Descifrado del mensaje con la llave privada de Pedro. Sólo Pedro puede descifrar el mensaje enviado por María ya que sólo él conoce la clave privada correspondiente.

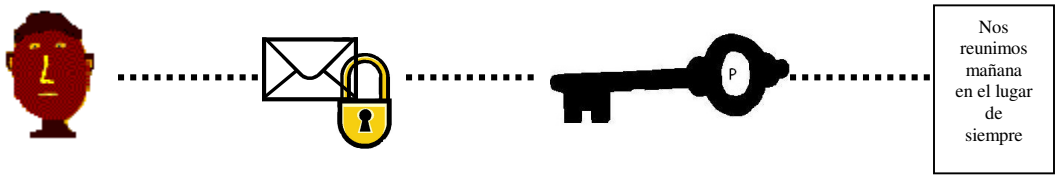


Figura 5.2 Descifrado del mensaje con la llave privada de Pedro

El beneficio obtenido consiste en la supresión de la necesidad del envío de la clave, siendo por lo tanto un sistema más seguro. El inconveniente es la lentitud de la operación.

5.5 Sistemas de cifrado híbridos

Criptográficamente, híbrido, se refiere a la combinación de cifrados simétricos y asimétricos, la velocidad de cifrado de los algoritmos de cifrado simétricos es acompañada de la seguridad de los algoritmos de cifrado asimétrico tales como el RSA, los métodos de cifrado como el de César, que son fáciles de romper, no son utilizados, en su lugar se utiliza el algoritmo IDEA, el cual es considerado un algoritmo simétrico seguro. Como ya se explicó antes IDEA fue desarrollado en 1991 por el profesor suizo James Massey y su estudiante Xuejia Lai, sus llaves están compuestas por 128 bits. Tales llaves son comúnmente cifradas utilizando RSA.

5.5.1 Cifrado con sistemas de cifrado híbridos

Para un mensaje que deba ser cifrado:

1. Se genera una llave secreta de IDEA cuya longitud es de 128 bits
2. Utilizando esta llave el mensaje es cifrado de una forma rápida
3. La llave es cifrada con la llave pública del destinatario
4. La llave cifrada junto con el mensaje cifrado es enviado al destinatario
5. El destinatario tiene primero que descifrar la llave cifrada de IDEA usando su llave privada RSA
6. Se obtiene la llave IDEA la cual utiliza para descifrar el mensaje cifrado

5.5.2 El sistema híbrido de cifrado más importante: PGP

PGP que significa "Pretty Good Privacy", es el sistema híbrido de cifrado más importante. En realidad el sistema de cifrado más exitosamente utilizado, fue creado por una sola persona, el científico Phil Zimmerman, quien lo hizo público en 1991. Cuando Zimmerman escuchó por primera vez acerca de la criptografía de llave pública en 1977, trabajaba como programador y también, tenía otro trabajo del cual no recibía salario, "salvador del mundo" *(Savior of the World)*, esto queda expuesto en la introducción de PGP que se muestra a continuación. Zimmerman se preguntó:"¿porqué no implementar un sistema de llave pública en computadoras personales utilizando el algoritmo RSA?".

Parte de la introducción del programa PGP:

"Es personal, privado y no le interesa a nadie más que a ti. Podrías estar planeando una campaña política, discutiendo tus impuestos, teniendo una aventura ilícita, o quizá estés haciendo algo que te parece legal, pero que no lo es. Lo que sea, no quieres que tus documentos o correos electrónicos sean leídos por otra persona. No hay nada malo al defender tu privacidad. Privacidad es un pastel de manzana como la Constitución.

Tal vez piensas que tu correo electrónico es tan legítimo que el cifrado no esta garantizado. Si eres un ciudadano apegado a la ley y que no tiene nada que esconder, entonces ¿porqué no envías siempre tus cartas hechas a mano en formato de postal?, ¿porqué no te sometes a pruebas de anti doppin?, ¿porqué requieres una certificación de la policía para que entren en tu casa?, ¿estas tratando de esconder algo?, debes ser un subversivo o un vendedor de drogas si escondes tus cartas dentro de sobres, o tal vez

eres un paranoico. ¿Tienen los ciudadanos normales, apegados a la ley, la necesidad de cifrar sus correos electrónicos?.

Que tal si todos pensáramos que los ciudadanos honrados deberían utilizar postales para sus cartas, entonces, si alguien utilizara sobres, despertaría sospechas. Tal vez las autoridades abrirían sus cartas para encontrar qué es lo que se esta ocultando. Afortunadamente, no vivimos en ese tipo de mundo, porque todos protegen la mayoría de sus cartas con sobres. De manera que nadie levanta sospechas al defender su privacidad con un sobre. Análogamente, sería bonito si todos, rutinariamente cifrarán sus correos electrónicos, inocentes o no, de manera que no se levantaran sospechas por los mensajes cifrados. Piensa en esto como un tipo de solidaridad.

Hoy en día, si el gobierno quiere violar la privacidad de ciudadanos ordinarios, tiene que trabajar en interceptar y evaporar las cartas que hayan sido interceptadas y leídas, además de escuchar cualquier conversación telefónica. Este es un tipo de tareas intensas no prácticas a gran escala. Esto sólo es hecho en casos importantes donde parece necesitarse.

Más y más de nuestras comunicaciones privadas viajan a través de canales electrónicos. El correo electrónico esta reemplazando, gradualmente, al correo convencional. Los mensajes de correo son demasiado fáciles de interceptar.

Nos vamos moviendo hacia un futuro donde la nación será atravesada con fibra óptica de gran capacidad, utilizada para comunicación de redes. El correo electrónico será una norma para todos, no la novedad que es ahora. El gobierno protegerá nuestra información con protocolos de cifrado diseñados por ellos mismos. Probablemente la mayoría de personas serán sumisas con respecto a ello. Pero tal vez algunas personas preferirán tener sus propias medidas de seguridad.

En 1992, una propuesta fue presentada al Congreso, en ella el FBI planteaba que todos las empresas dedicadas a la fabricación de equipo de comunicaciones construyeran una un tipo especial de puerto que permitiría al FBI interceptar todo tipo de comunicación electrónica. Aunque no logró apoyo alguno, debido a la oposición de los ciudadanos, ésta fue presentada nuevamente en 1994.

Pero, lo más alarmante de todo fue la nueva iniciativa para el cifrado de la información de la Casa Blanca, bajo el desarrollo de la NSA desde el inicio de la administración de George Bush, la cual se hizo pública en abril de 1994. La iniciativa consistía en un dispositivo de cifrado llamado Clipper chip, el cual contenía un nuevo algoritmo de cifrado, clasificado, desarrollado por la NSA. El gobierno animaba a las empresas a implementar el dispositivo en sus propios dispositivos de comunicación como por ejemplo, los teléfonos. La empresa AT&T esta instalando Clipper en sus dispositivos de comunicación. La idea principal: al momento de fabricación cada circuito del Clipper será configurado con una llave única, de la cual el gobierno deberá tener una copia. Pero, no hay de qué preocuparse: el Gobierno promete utilizar las llaves para leer el tráfico sólo cuando este debidamente autorizado para hacerlo.

Agencias de inteligencia, las empresas petroleras y otros gigantes corporativos tienen acceso total a la mejor tecnología de cifrado, pero las personas ordinarias no.

PGP permite a las personas tomar la privacidad por sus propias manos. Existe una creciente necesidad de ello, esa es la razón por la que lo creé".

Phil Zimmerman

5.5.3 Cifrado utilizando sistemas de cifrado híbrido

El uso de llaves asimétricas hace lento el proceso de cifrado. Para solventar dicho inconveniente, el procedimiento que suele seguirse para realizar el cifrado de un mensaje es un método híbrido, consistente en utilizar un algoritmo de llave pública junto a uno de llave simétrica.

María y Pedro tienen sus pares de llaves respectivas.

María escribe un mensaje a Pedro. Lo cifra con el sistema de criptografía de llave simétrica. La llave que utiliza se llama llave de sesión y se genera aleatoriamente.

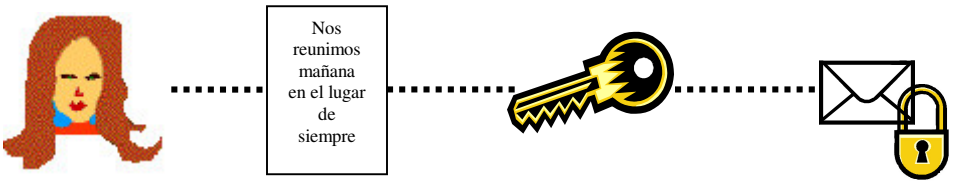


Figura 5.3 Mensaje de María para Pedro utilizando un algoritmo de llave simétrica

Para enviar la llave de sesión de forma segura, esta se cifra con la llave pública de Pedro, utilizando por lo tanto criptografía de llave asimétrica.

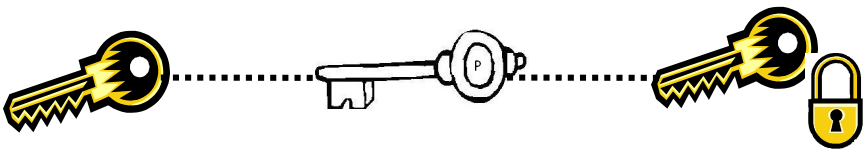


Figura 5.4 Cifrado de la llave de sesión utilizando un algoritmo de llave asimétrica

Pedro recibe el mensaje cifrado con la llave de sesión y ésta misma cifrada con su llave pública.

Para realizar el proceso inverso, en primer lugar utiliza su llave privada para descifrar la llave de sesión.

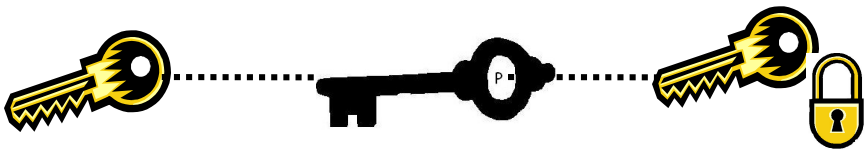


Figura 5.5 Descifrado de la llave de sesión

Una vez ha obtenido la llave de sesión, ya puede descifrar el mensaje.

Figura 5.6 Descifrado del mensaje

Con este sistema se obtiene:

Confidencialidad: sólo podrá leer el mensaje el destinatario del mismo.

Integridad: el mensaje no podrá ser modificado.

Pero todavía quedan sin resolver los problemas de autenticación y de no repudio. Los cuales se solventan con la utilización de las firmas digitales, las cuales se explican posteriormente en este capítulo.

5.6 Funciones resumen (Hash)

Cuando los mensajes a firmar son muy largos es una práctica común la de firmar un resumen del mensaje en lugar de firmar el mensaje completo. El resumen se obtiene mediante procedimientos de compresión de información a través de funciones resumen o funciones Hash.

Matemáticamente las funciones resumen se definen como proyecciones de un conjunto, generalmente con un número elevado de elementos (incluso infinitos), sobre un conjunto de tamaño fijo y mucho más pequeño que el anterior, es decir, transforman un mensaje de m bits en otro resumen de n bits, siendo $n < m$.

Las funciones resumen se utilizan no solamente para reducir las necesidades de almacenamiento sino también por razones prácticas, para agilizar el proceso de generación y verificación de firmas digitales.

La integridad de los datos es crucial para cualquier sistema de seguridad. Al utilizar los resultados generados por las funciones resumen se puede detectar cambios no autorizados en archivos.

Características de las funciones resumen

Dada una función resumen $y = H(x)$, ésta deberá cumplir con las siguientes condiciones:

1. La función $y = H(x)$ es de una sola dirección, es decir, el cálculo del valor $y = H(x)$ es sencillo, pero el cálculo de $x = H^{-1}(y)$ es tan complejo que no se puede llevar a cabo con los conocimientos matemáticos actuales, ni siquiera disponiendo de las más avanzadas capacidades de cálculo.

2. Dado un valor x, es computacionalmente imposible encontrar otro valor $x' \neq x$ tal que $H(x') = H(x)$.

3. Dado el valor H (x), es computacionalmente imposible encontrar otro valor $x' \neq x$ tal que $H(x') = H(x)$.

4. La función $y = H(x)$, es libre de colisiones, es decir, es computacionalmente improbable encontrar una pareja de valores (x', x) con $x' \neq x$ tales que $H(x) = H(x')$.

   Se puede definir una colisión como: $h(x_1) = h(x_2)$, ésta ocurre cuando dos mensajes distintos producen un mismo resultado. Es importante aclarar que cada función resumen posee un número infinito de estas colisiones, de manera que, una función resumen nunca podrá estar libre de colisiones. Y, el requerimiento de la resistencia a las colisiones apunta a que a pesar que las colisiones existan, estas no deberán ser encontradas.

Las funciones resumen se pueden clasificar en dos grandes grupos:

- *Funciones resumen con clave*. Estas funciones tiene como uno de sus argumentos la clave del sistema de cifrado utilizado para transmitir la información.

- *Funciones simétricas sin clave*. Estas funciones tiene como argumento tan sólo el mensaje a compactar, al cual le aplican técnicas de compresión con las características anteriormente indicadas.

- Dentro de éste último subgrupo se pueden a su vez distinguir dos tipos diferentes de funciones resumen:

- Las funciones OWHF *(One Way Hash Functions)*, también denominadas funciones resumen *débiles*.

- Las funciones CRHF *(Collision Resistant Hash Functions)*, también denominadas funciones resumen *robustas*.

La diferencia entre ambas es que las primeras (débiles) satisfacen únicamente las tres primeras condiciones apuntadas anteriormente, mientras que las segundas (robustas) son además libres de colisiones.

A diferencia de los algoritmos de cifrado de bloques, las funciones resumen no utilizan llaves, de manera que no existe un ataque que implique la búsqueda de una llave. El parámetro a utilizar es entonces, la longitud de mensaje de salida.

Existen muy pocas funciones resumen que sean efectivas, además, parecería que la criptografía se ha estancado en la familia de SHA y MD5, las cuales aunque no han sido suficientemente analizadas ya han sido estandarizadas por la NIST.[77]

A continuación se presentan algunas de las funciones resumen más conocidas:

[77] Ferguson, N., Schneier, B. Op. Cit. pp. 84.

5.6.1 MD4 – Message Digest 4

La MD4 es una función de un solo sentido diseñada por Ron Rivest, ésta produce un resultado de 128 bits a partir del mensaje de entrada.

Después que el algoritmo fue presentado, los criptógrafos Bert den Boer y Antoon Bosselaers criptoanalizaron las dos últimas vueltas, de las tres que el algoritmo ejecuta. Posteriormente, en un ataque no relacionado con el resultado antes mencionado, Ralph Merkle atacó las dos primeras vueltas. Aunque estos ataques no se extendían al algoritmo completo, Ron Rivest lo fortaleció y el resultado fue un nuevo algoritmo al que llamó MD5.

5.6.2 MD5 – Message Digest 5

En 1991 Ron Rivest desarrolló la función MD5 que no es más que una versión mejorada de MD4, aunque es más compleja, su resultado, al igual que su antecesora es de 128 bits. Consta de cuatro vueltas y en cada una de ellas el mensaje es mezclado utilizando operaciones como suma, XOR, AND, OR y la rotación de palabras de 32 bits.

Una de las funciones más utilizada en criptografía es la SHA (Secure Hash Algorithm), la cual se describe a continuación.

5.6.3 El SHA como estándar

La primera versión, SHA (Secure Hash Algorithm), fue propuesta en 1992 por el NIST, como estándar federal para los Estados Unidos, para su utilización en firmas digitales.
El algoritmo SHA se considera seguro, basados en que es computacionalmente imposible recuperar el mensaje correspondiente a un determinado resumen e igualmente es muy poco probable encontrar dos mensajes diferentes con el mismo resumen. Es decir que la función SHA es de una sola dirección.

El SHA-1 tiene varias propiedades en común con MD5, a pesar de tener un diseño más conservador, y ser dos veces más lento que MD5.

5.6.4 Seguridad del algoritmo SHA

El algoritmo SHA es básicamente igual al MD5. Los principales cambios del SHA respecto a MD5 son la inclusión de una transformación de expansión, una etapa extra de procesado y la adición de cada uno de los pasos de la salida del paso anterior para lograr un efecto avalancha mejorado. Por su parte, el algoritmo MD5 es básicamente igual al MD4 con una etapa extra de procesado y una mejora en el efecto avalancha.

Los algoritmos de las funciones SHA-1 y MD5 son considerados seguros debido a que no existen técnicas conocidas para encontrar colisiones, a excepción de la fuerza bruta. En un ataque de fuerza bruta, varias entradas aleatorias son probadas, almacenando los resultados hasta que alguna colisión es encontrada. Se puede esperar encontrar una colisión entre las 2n/2 operaciones, donde n es el número de bits del resultado generado por la función, este ataque es comúnmente conocido como el ataque del cumpleaños o birthday attack[78]. Lo anterior quiere decir que un atacante necesitaría probar los resultados de aproximadamente 264 mensajes para encontrar una colisión

[78] Ver Anexo I.

en la función resumen MD5, y aproximadamente 280 cálculos para encontrar una colisión en SHA-1.

5.6.5 SHA-256, SHA-384 y SHA-512

El NIST publicó un documento que contenía tres nuevas funciones resumen[79], ellas tienen bits de entrada de 256, 384 y 512 respectivamente. Y están diseñadas para ser utilizadas con las llaves de 128,196 y 256 bits del AES. La estructura de estos nuevos algoritmos es similar a la del SHA-1.

Estas funciones resumen son nuevas, y no se recomienda utilizarlas. Aunque si se quiere más seguridad que la brindad por SHA-1 entonces se necesita una función resumen con un resultado mayor, aunque ninguna de ellas ha tenido publicaciones en las cuales es les analicen.

La función SHA-256 es mucho más lenta que SH-1, para mensajes largos el cálculo con esta nueva función resumen toma el mismo tiempo o un poco más que si se utilizara AES o Twofish.

La función SHA-384 es relativamente inútil. Para ejecutarla, se necesita hacer todo el trabajo que es requerido para ejecutar la SHA-512, y luego deshacerse de algunos de los bits.

5.7 ¿Cuál función resumen se debe utilizar?

No existen muchas opciones, pero, la recomendación es utilizar una de la familia SHA. A diferencia de los cifrados de bloque, muy poco se ha investigado acerca de las funciones resumen.

Si existiera una función resumen perfecta que pudiera ser ejecutada rápidamente y que además su seguridad no pudiera ser violada, ni en un millón de años, nadie tendría que decidir cual función resumen utilizar. Pero, este no es el caso, y para ayudar a tomar la decisión, existen dos factores que se deberán tomar en cuenta:

MD5 - Mayor velocidad y menos seguridad
SHA-512 Cuando la seguridad es el aspecto más importante

Las especiales características de las funciones resumen permiten que éstas puedan ser utilizadas por cualquier firma digital para garantizar la autenticidad de contenido de los mensajes firmados. Esto no es estrictamente necesario, ya que la firma digital de un mensaje con la llave secreta del firmante es suficiente para demostrar la autenticidad del contenido del mismo. Esto es debido a que sí el mensaje fuera modificado su firma también habría de serlo y ello requeriría el conocimiento de la clave secreta del firmante.

5.8 Firmas digitales

Partiendo de la necesidad de comunicación entre dos personas, cuyo único medio de comunicación sea el Internet, uno de los problemas más comunes es ¿cómo saber que efectivamente la persona con quien se establece la comunicación es la quien dice ser?. Es entonces cuando surge el problema de verificación de Identidad o autenticación, los casos en los que este problema se presenta con mayor frecuencia son:

[79] Federal Information Processing Standards Publication 180-2, 2002 August 1, *Announcing the Secure Hash Standard.*

- Dos personas se ven en la calle y éstas ya se conocen, se saludan con sus nombres, cada una de ellas verifica la identidad de la otra visualmente y aceptan que es la persona que ya conocen.
- Dos personas que se conocen pero que no se ven, por ejemplo que esta una del lado de una puerta y la otra del otro lado, ¿cómo pueden reconocerse?. Lo más fácil es hacer preguntas y dependiendo de las respuestas aceptar la identidad de la otra persona.
- Dos personas tienen que validar su identidad pero no se conocen anteriormente, pero pueden comprobar que son quien dicen ser mediante documentos de identidad personal que incluyan una fotografía.

Existen casos en los que es necesario verificar la identidad, en este caso a veces será necesario quedarse con un comprobante de tal verificación, por ejemplo al cobrar un cheque en un banco, el cajero pide un documento de identidad, realiza una identificación visual de acuerdo al sujeto que desea cobrar el cheque y la fotografía del documento presentado, además solicita que el cheque este firmado.

La firma tradicional tiene varias características, la principal de ellas es que es aceptada legalmente, esto quiere decir que si alguna persona firmó un documento adquiere tanto los derechos como las obligaciones que de él deriven, y si estas obligaciones no son acatadas el portador del documento tiene el derecho de reclamación mediante un litigio. La autoridad competente acepta las responsabilidades adquiridas con sólo calificar a la firma como válida.

Existen dos procedimientos importantes, el primero el proceso de firma, que es el acto cuando una persona "firma" manualmente un documento. Y el proceso de verificación de la firma, que es el acto que determina si una firma es válida o no.

Proceso de Firma: consiste sólo en tomar un bolígrafo y estampar, dibujar o escribir garabatos en un papel. En general este garabato debe ser el mismo y es elegido a gusto de la persona. Se usa como una marca personal.

Proceso de verificación: existen en general dos métodos de verificación de la firma, uno es el más usado y simple, que es el visual, esté método lo aplica cualquier cajero al pagar un cheque, o al efectuar un pago con tarjeta de crédito.

En la práctica la criptografía simétrica y asimétrica se usan conjuntamente. La simétrica para intercambiar grandes volúmenes de información por su rapidez. Y la asimétrica para el intercambio de las claves simétricas y la firma digital.

Ya se ha mencionado el principal inconveniente de los algoritmos de llave pública: su lentitud que, además, crece con el tamaño del mensaje a cifrar. Para evitar éste problema, la firma digital hace uso de funciones resumen.

Las firmas digitales son análogas a las firmas analógicas tradicionales de los documentos escritos sobre papel. Estas últimas están basadas en la forma física en que una persona firma su nombre. Las firmas analógicas son siempre similares y fáciles de falsificar.

Las firmas digitales deben de ser diferentes para cada documento. Cada firma debe depender de cada documento en el que se utilice, a través de una relación matemática, de forma que esta relación permita posteriormente, verificar la validez de la firma y con ello la autenticidad de origen e integridad del documento firmado.

La imposibilidad de falsificar cualquier tipo de firma radica en el secreto del firmante. En el caso de las firmas analógicas el secreto está constituido por características de tipo grafológico inherentes al firmante y por ello son difíciles de falsificar. En el caso de las firmas digitales, el secreto del firmante es el conocimiento exclusivo de una llave secreta utilizada para generar la firma.

Para garantizar la seguridad de las firmas digitales es necesario que éstas sean:

- *Únicas*: Las firmas deben poder ser generadas solamente por el firmante.

- *Infalsificables*: Para falsificar una firma digital un delincuente tiene que resolver problemas matemáticos de una complejidad muy elevada, es decir, las firmas han de ser computacionalmente seguras.

- *Verificables*: Las firmas deben ser fácilmente verificables por los receptores de las mismas.

- *Innegables*: El firmante no debe ser capaz de negar su propia firma.

- *Viables*: Las firmas han deben ser fáciles de generar por parte del firmante.

Diversos protocolos para firmas digitales, basados en la criptografía simétrica fueron propuestos y utilizados, pero los basados en la criptografía de llave pública demostraron su superioridad conceptual y operacional.[80]

[80] Pastor, J., Sarasa, M. Op. Cit. Pp. 124.

Ejemplo de utilización de firmas digitales:

María escribe un mensaje a Pedro. Es necesario que Pedro pueda verificar que realmente es María quien ha enviado el mensaje. Por lo tanto María debe enviarlo digitalmente firmado, para lo cual realizará los siguientes pasos:

1. Aplica una función resumen al mensaje a enviar.

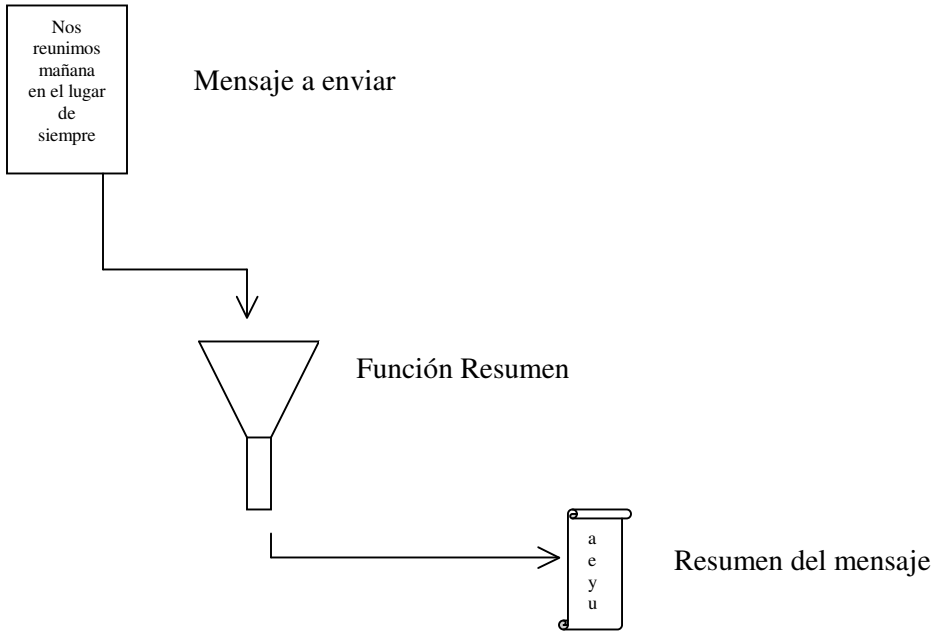


Figura 5.7 Aplicación de la función resumen

2. Cifra el resultado de la función resumen con su clave privada. De esta forma obtiene su firma digital.

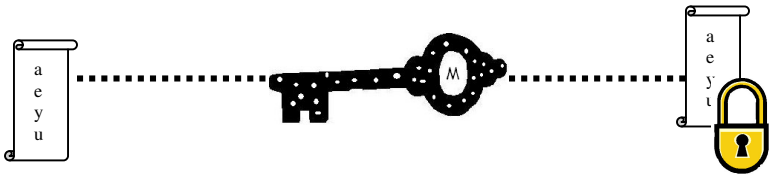


Figura 5.8 Generación de la firma digital

3. María envía a Pedro el mensaje original junto con la firma.

Figura 5.9 Envío del mensaje junto a la firma digital

Pedro recibe el mensaje junto a la firma digital. Deberá comprobar la validez de ésta para dar por bueno el mensaje y reconocer al autor del mismo (integridad y autenticación).

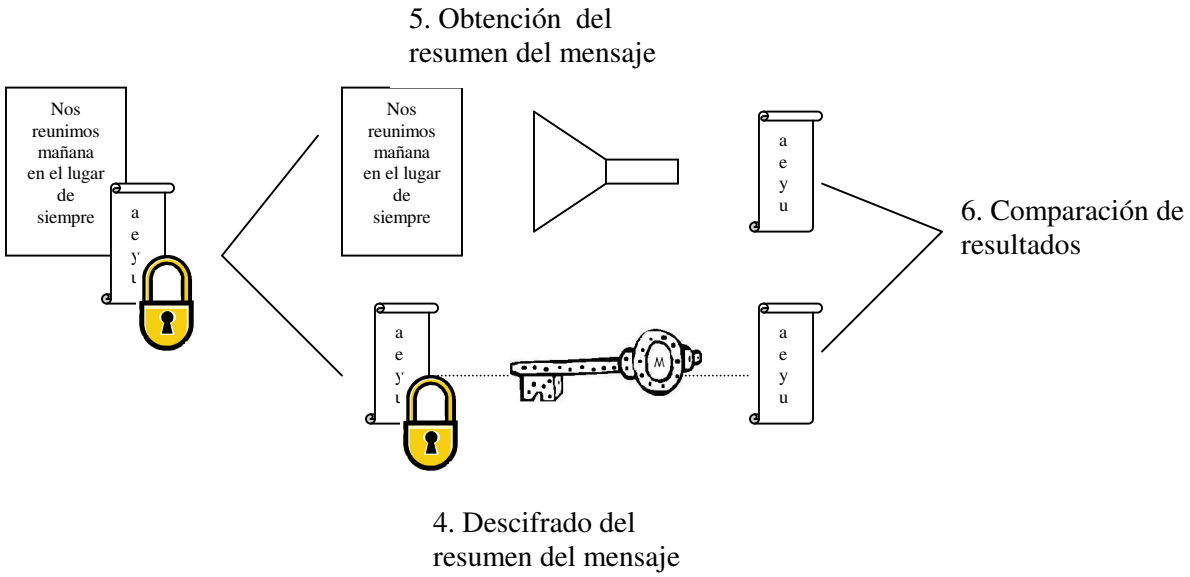


Figura 5.10 Descifrado del mensaje y comprobación de la firma digital

4. Se descifra el resumen del mensaje mediante la clave pública de María

5. Se aplica al mensaje la función hash para obtener el resumen.

6. Se compara el resumen recibido con el obtenido a partir de la función hash. Si son iguales, Pedro puede estar seguro de que quien ha enviado el mensaje es María y que éste no ha sido modificado.

Con este sistema se obtiene:

Autenticidad: la firma digital es equivalente a la firma física de un documento.

Integridad: el mensaje no podrá ser modificado.

No repudio de origen: el emisor no puede negar haber enviado el mensaje.

5.9 PGP Pretty Good Privacy

Este procedimiento fue desarrollado en el año de 1991 por Phil Zimmerman y es accesible de forma gratuita a través del Internet. En concreto el método PGP garantiza:

- Confidencialidad: el mensaje puede ser leído sólo por receptores autorizados.
- Autenticidad de origen: permite verificar la identidad del emisor del mensaje
- Integridad: garantiza que el mensaje no ha sido modificado
- No repudio de origen: permite la verificación de la identidad del emisor original de un determinado mensaje.

Este algoritmo utiliza tanto algoritmos de clave secreta como de llave pública. Los algoritmos de clave secreta se utilizan para el cifrado de los datos, mientras que los de llave pública son utilizados para la administración de las llaves, también, utiliza funciones resumen de una sola dirección en el cálculo de las firmas digitales. Los algoritmos de cifrado más comúnmente utilizados por el PGP se indican a continuación:

- *Cifrado de datos:*
  IDEA en modo CBC
- *Autenticidad de origen y autenticidad de contenido:*
  RSA y MD5
- *Administración de llaves*
  RSA

La longitud de las claves RSA utilizadas por el PGP varía de acuerdo con el nivel de seguridad, pudiendo ser de 384, 512 y 1024 bits.

El propósito fundamental del PGP es el de enviar mensajes de forma segura. En general, éstos se envían cifrados y firmados. Sin embargo, también es posible cualquier combinación de los parámetros anteriores, es decir, se pueden enviar mensajes únicamente cifrados o únicamente firmados. Los mensajes cifrados pueden ser leídos sólo por sus receptores autorizados, mientras que cualquiera puede verificar la integridad del mensaje en claro firmado con sólo utilizar la clave pública del firmante. Además, puesto que la gestión de claves se lleva a cabo mediante un protocolo de clave pública, el emisor y el receptor pueden intercambiar la clave secreta de cifrado del mensaje sin necesidad de utilizar un canal de comunicación seguro.

Todas las estructuras de datos utilizadas para la transmisión de mensajes con PGP son de tipo binario, lo cual hace que ésta pueda llevarse a cabo de manera bastante eficiente. Por esta razón, los mensajes de PGP no tienen una estructura transparente para el usuario.

Un mensaje PGP consiste en la concatenación de uno o más paquetes, entendiendo como tales a bloques digitales de información con un determinado formato y características. Cada paquete consta básicamente de tres partes, la primera almacena la información relativa al tipo de paquete, la segunda, indica su longitud y la tercera contiene la información del paquete propiamente dicha. Una característica importante de esta estructura es que puede anidarse, es decir, un paquete puede incluirse dentro de otro.

La certificación de llaves en PGP está basada en la idea de que la confianza es un concepto social: la gente confía en sus amigos. Así, en PGP la llave pública de cada usuario está firmada por aquellos, que confían en él, siendo precisamente estas firmas las que certifican la validez de dicha llave otros usuarios. De esta forma, un determinado usuario puede confiar en la validez de la llave

de otro siempre y cuando dicha llave esté firmada por alguien en quien el primer usuario confía, es decir, si existe al menos un amigo común a ambos.

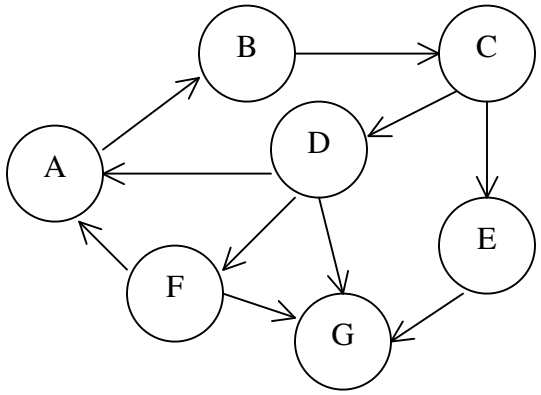


Figura 5.11 Estructura de certificación del PGP

Así, por ejemplo, el usuario A puede confiar en la validez de la llave pública del usuario C, puesto que está certificada por el usuario B, en quien A confía. Sin embargo, el usuario E no puede confiar en la validez de la clave pública de ningún otro usuario a excepción de G, cuya llave certifica. Por lo tanto, el principal inconveniente de este procedimiento de certificación es que un determinado usuario no siempre puede verificar la validez de la llave de otro, a menos que ambos tengan como mínimo un amigo común.

5.9.1 Debilidades de PGP

- *Debilidades de implementación*: agujeros de seguridad provocados por una implementación defectuosa de PGP, y corresponden a versiones concretas del programa. Por ejemplo, el fallo descubierto en la versión 5.0 de PGP para UNIX, que hacía que las llaves no fueran aleatorias, o el encontrado en todas las versiones para Windows, desde la 5.0 hasta la 7.0.4, en la que un inadecuado procesamiento de los códigos ASCII permitía a un atacante introducir archivos en la computadora de la víctima.

- *Intrínsecas del protocolo:* a principios del año 2001 se hizo pública una técnica que permitía a un atacante falsificar firmas digitales. En este caso se necesitaba el acceso físico a la computadora de la víctima para manipular su llave privada, por lo que la falla carece de interés práctico.

CONCLUSIONES

La protección de la información es un asunto que en ciertos ámbitos es de vital importancia. Esto ha conducido a lo largo de la historia a la búsqueda de mecanismos que permitan proveer seguridad a la comunicación. Entre los diferentes recursos de los que se ha hecho uso, la criptografía ocupa una relevante posición, si bien no necesariamente la más preponderante en cada caso. En general, la seguridad suele brindarse a través del establecimiento de políticas que incluyen determinado tipo de recursos en función de las condiciones y la naturaleza de la información a proteger.

Se consideran fundamentales para la criptografía los criterios de confidencialidad, autenticidad e integridad. Con el paso del tiempo se han ido agregando otras características deseables y necesarias en los sistemas criptográficos, pero éstas siguen considerándose básicas.

La evolución histórica de la criptografía puede ser segmentada en tres etapas que presentan diferencias sensibles entre sí: una artesanal, una técnica y una digital. La primera abarca un largo período de tiempo que surge en tiempos ancestrales y llega hasta fines del siglo XIX y principios del XX; se caracterizó por el empirismo en la creación y uso de los métodos. El inicio de la segunda se traslapa con el fin de la primera y concluye en los años 40; se caracterizó por la mecanización de métodos que en su momento eran útiles pero a la vez engorrosos para operadores que tuvieran que ejecutarlos manualmente; otro aspecto que caracterizó a esta era fue la relativa escasez de aportes teóricos significativos, aunque vale la pena resaltar que a ella pertenece la idea de Kerckoffs de que el algoritmo puede ser público mientras la llave permanezca oculta. La tercera surge como resultado de los trabajos de investigadores como Turing y Shannon; en sus inicios hubo una fuerte teorización que buscaba dar soporte a trabajos de criptografía y criptoanálisis para fines primordialmente militares. Pertenecen a ella el desarrollo de sistemas criptográficos completamente electrónicos, el establecimiento de estándares criptográficos para diferentes fines y la teorización sobre nuevas formas de llevar a cabo tareas de cifrado digital de datos.

Como es de esperar, en la actualidad todos los métodos son digitales. Hay dos grandes familias de métodos de cifrado, los de llave privada y los de llave pública.

Para un grupo pequeño de personas, el intercambio de sus llaves privadas, es una tarea fácil, sin embargo, a medida que el grupo crece, la administración de las llaves se complica para poder intercambiarlas de forma segura deberán reunirse y cada uno hacer entrega de su llave, situación que deberán repetir cada vez que alguno de ellos actualice o cree una nueva. Es entonces cuando nos encontramos con las desventajas de la utilización de llaves privadas: su distribución y manejo.

Por ejemplo, para dos personas que necesitan comunicarse en secreto, ellos tendrán que acordar una llave que nadie más conozca. Si n personas que necesitan comunicarse, el número de llaves necesarias serían $n(n-1)/2$. Por ejemplo, para 100 personas que se desean comunicar en secreto, se requerirían 4,950 llaves que cada usuario tendría que resguardar.

Cuando una llave es accidentalmente revelada, automáticamente toda la información que haya sido cifrada con ella quedará expuesta ante cualquier persona que la conozca; es de acá que surge la recomendación de utilizar una llave diferente para cada cifrado, lo que implica almacenar o memorizar gran cantidad de llaves, y esto también nos lleva a deducir que la vida de la llave es muy corta, ya que sólo se debe utilizar para cifrar una vez.

Por otra parte, con las llaves privadas se utilizan los secretos compartidos, es decir, sólo el emisor y el receptor deberán conocer la llave y comprometerse a mantenerla en secreto. De manera que la robustez de estos algoritmos recae sobre el conocimiento de dicha llave.

Ante la revelación de la llave, no habrá manera de evitar que una persona se haga pasar por el emisor o receptor, peor aún, no habrá manera de comprobar su identidad. Lo antes mencionado abre un gran agujero de seguridad, y nos recuerda que los secretos mejor guardados son aquellos que no se comparten.

El DES fue el estándar durante alrededor de treinta años, pero debido a ciertas debilidades que se le fueron detectando (principalmente la longitud de la llave y la estructura de las cajas de sustitución), en 1997 el NIST convocó a un concurso para la elección de un nuevo estándar de cifrado, de las quince propuestas recibidas se eligieron cinco finalistas, de los cuales Rijndael fue declarado como el nuevo AES, ya que era el que mejor combinaba características como: seguridad, rendimiento, eficiencia, facilidad de implementación y flexibilidad en cuanto a tamaño de bloque y llave.

El reemplazo del DES por el AES se ha realizado paulatinamente y, en muchos casos, se sigue utilizando el DES combinado con algoritmos de llave pública, como es el caso de las transacciones con tarjetas de crédito, telecomunicaciones y codificación de señales satelitales.

Se debe tener claro que las debilidades del DES no se deben a un diseño deficiente, sino al tiempo que transcurrió desde su creación y al avance que las técnicas de criptoanálisis y la potencia de procesamiento que las computadoras han adquirido, lo que nos hace pensar que ocurrirá lo mismo con el AES, pero, podemos estar tranquilos ya que éste último es relativamente nuevo y a la fecha no se le ha conocido ningún tipo de ataque exitoso.

En este trabajo se procedió a escribir varios de los algoritmos de cifrado simétrico de bloques más importantes, posteriormente se hizo una comparación entre ellos a partir de criterios de velocidad, facilidades de implementación en hardware y/o software y vulnerabilidad. Esta comparación corrobora algunas de las condiciones de diseño más importantes que permitieron que el algoritmo Rijndael se eligiera como nuevo estándar.

Lo anterior no descalifica como posibles alternativas en ciertas aplicaciones concretas a algunos de los otros algoritmos. Hay algunos que tienen niveles de vulnerabilidad relativamente bajos, velocidades razonablemente altas y requieren condiciones de implementación relativamente simples. Tales son los casos de IDEA y Blowfish, sin embargo, hay que resaltar el hecho de que IDEA esta patentado y requiere licencia para su uso comercial, en tanto que Blowfish es abierto.

Mención especial merece el caso de Skipjack que está patentado y cuya implementación únicamente está autorizada para hardware, por lo que aunque es bastante difundido en cierto tipo de aplicaciones sólo hay disponibles implementaciones en circuitos integrados cuyo precio no suele ser elevado.

Como solución al problema de la distribución de llaves nació la criptografía de llave pública, que se basa en la utilización de dos llaves, una para cifrar y otra para descifrar.

El RSA es el algoritmo más utilizado, y su seguridad se basa en la dificultad de factorizar dos números grandes. Este algoritmo busca que dos personas puedan establecer comunicación sin tener que entregar la llave privada con anterioridad; al contario de lo que ocurre con los algoritmos de llave privada, su robustez no recae sobre el conocimiento de una sola llave.

Cada persona deberá tener únicamente dos llaves, una pública, que como su nombre indica será conocida públicamente y una privada, sin que el conocimiento de la primera lleve al descubrimiento de la última. La desventaja es que al hacer que cada persona maneje dos llaves el sistema se vuelve lento y complejo.

Con estos algoritmos el único secreto que se deberá guardar es el de la llave privada, lo que deja de lado los secretos compartidos utilizados con los algoritmos de llave privadas.

Al haber una llave pública y otra privada, es factible la autenticación del mensaje y el emisor.

Debido a la complejidad de cálculo que realizan los algoritmos de llaves privadas, éstos son mucho más lentos que los de llave pública, es por ello que aprovechando la velocidad de procesamiento de los algoritmos de llaves privadas, y la seguridad de los de llaves públicas, se utiliza una mezcla de ellos y surgen así los sistemas de cifrado híbridos, que emplean la rapidez de los algoritmos de llave privada y la gestión de llaves de los de llave pública.

BIBLIOGRAFÍA

[1] Anderson, R. *Two remarks on Public Key Cryptology.* Computer Laboratory. University of Cambridge. Cambridge, United Kingdom. S/f.

[2] Bauer, F. Decrypted Secrets. Springer Verlag. New York. 2002.

[3] Biham, E., Shamir, A. *Differential Cryptanalysis of the Data Encryption Standard.* Proceedings, Berlin. Springer Verlag. 1993.

[4] Biham, E., Shamir, A. *Differential Cryptanalysis of DES-like Cryptosystems* (Extended Abstract). The Weizmann Institute of Science. Department of Applied Mathematics. S/f.

[5] Biham, E., Shamir, A. *Differential Cryptanalysis of Snefru, Khafre, REDOC-II, LOKI and Lucifer*. (Extended Abstract). The Weizmann Institute of Science. Department of Applied Mathematics and Computer Science. Rehovot. S/f.

[6] Biham, E. *New Types of Cryptanalytic Attacks Using Related Keys.* In Tor Helleseth, editor, Advances in Cryptoglogy –EUROCRYPT´93, volume 765 of Lecture Notes in Computer Science, pages 398-409. Springer-Verlag. 1993.

[7] Biryukov, A. *Block Ciphers and Stream Ciphers: The State of the Art*. Katholieke Universiteit Leuven. Dept. ESAT/SCD-COSIC. Leuven. S/f.

[8] Biryukov, A., Nakahara, J., Preneel, B., Vandewale, J. *New Weak-Key Clases of IDEA*. Katholieke Universiteit Leuven. Dept. ESAT/COSIC. Leuven. s/f.

[9] Blelloch, G. *Introduction to Cryptography*. Rearrangements and adaptation of the Tzu-Yi Chen, David Oppenheimer and Marat Boshernitsan lecture notes (Fall 1997). October 2000.

[10] Bresson, E., Chevassut, O., Poitcheval, D. *Provably Authenticated Group Diffie-Hellman Key Exchange – The Dynamic Case*. (Full Version). Advances in Cryptology – Proceedings of Asiacrypt 2001. 9-12 December. Gold Coast (Australia).

[11] Brisson, R., Théberge, F. Un aperçu de l'histoire de la cryptologie. Descargado de http://collection.nlc-bnc.ca/100/200/301/cse-cst/overview-f/musee.pdf el 7 de diciembre de 2003.

[12] Churchhouse, R. Codes and Ciphers. Julius Caesar, the Enigma and the Internet. Cambridge University Press. Cambridge. 2002.

[13] CLE International. Dictionaire du Français. Dictionaires LeRobert. CLE International. Paris. 1999.

[14] Daemen, J., Knudsen, L., Rijmen, V. *The Block Cipher Square*. Fast Software Encryption, LNCS 1267, E. Biham, Ed., Springer-Verlag, 1997, pp. 149-165. Also available as http://www.esat.kuleuven.ac.be/rijmen/square/fse.ps.gz.

[15] Daemen, J., Rijmen, V. *AES Proposal: Rijndael*. Document Version 2, Date: 03/09/99.

[16] Dandalis, A. Prasanna, V. Rolim, J. *A Comparative Study of Performance of AES Final Candidates Using FPGAs*. Workshop on Cryptographic Hardware and Embeddes Systems, August 2000.

[17] Delahaye, J. *La cryptographie RSA vingt ans après*. Pour la Science. No. 267. Janvier 2000. pp. 104-108.

[18] Diffie, W., Hellman, M. *New Directions in Cryptography*. IEEE Transactions on Information Theory. IT-22(6). November 1976.

[19] Diffie, W. *The First Ten Years of Public-Key-Cryptography*. IEEE Publication. New York, 1992.

[20] Dubertret, G. Initiation à la cryptographie. Éditions Vulbert. Paris. 2002. 3ème. Édition.

[21] Ferguson, N., Schneier, B. Practical Cryptography. Wiley Publishing Inc. Indianapolis. 2003.

[22] González, M., Shparlinski, I. *On the Security of Diffie-Hellman Bits*. Preprint 2000.

[23] Herodoto. Los nueve libros de la historia. Clásicos Jackson. México DF. 1966.

[24] Herodoto. Los nueve libros de la historia. CONACULTA-Océano. México DF. 1999.

[25] Heys, H. *A Tutorial on Linear and Differential Cryptanalysis*. Electrical and Computer Engineering. Faculty of Engineering and Applied Science. Memorial University of Newfoundland. Newfoundland. S/f.

[26] Heys, H., Tavares, S. *On the Design of Secure Block Ciphers*. Department of Electrical and Computer Engineering. Queen's University. Kingston, Ontario. S/f.

[27] Jamil, T. *The Rijndael Algorithm*. IEEE Potentials. Vol. 23, Issue 2, April-May 2004. pp. 36-38.

[28] Johnson, N., Jajodia, S. *Exploring Steganography: Seeing the Unseen.* Computer Practices. S/f. Pp. 26-34.

[29] Kahn, D. The Codebreakers. The Comprehensive History of Secret Communication from Ancient Times to the Internet. Scribner. New York. 1996.

[30] Kerckhoffs, A. *La cryptographie militaire.* Journal des Sciences Militaires. Janvier 1883.

[31] Lai, X., Massey, J. *A Proposal for a New Block Encryption Standard*. Reprint of pp. 389-404 in Advances in Cryptology-EUROCRYPT'90 Proceedings, LNCS 473, Springer-Verlag. 1991.

[32] Lucena, M. Criptografía y seguridad en computadores. Escuela Politécnica Superior. Universidad de Jaén. Jaén, España. 1999. 2ª Edición.

[33] Mel, H., Baker, D. Cryptography Decrypted. Addison-Wesley. Boston, 2001.

[34] Menezes, A., van Oorschot, P., Vanstone, S. Handbook of Applied Cryptography. CRC Press. 1996. Available from http://www.cacr.math.uwaterloo.ca/hac

[35] Mercier, D. Cryptographie classique. IUFM de Guadaloupe, Morne-Ferret. Poite-à-Pitre. 5 octobre 2003.

[36] Mirza, F. *Block Ciphers and Cryptanalysis*. Royal Holloway University of London. London. S/f.

[37] NIST. Federal Information Processing Standards Publication. 46, 1977, Data Encryption Standard.

[38] NIST. Federal Information Processing Standards Publication. 81, 1980, DES Modes of Operation.

[39] NIST. Federal Information Processing Standards Publication. 74, 1981, Guidelines for Implementing and Using the NBS Data Encryption Stardard.

[40] NIST. Federal Information Processing Standards Publication 180-2, 2002 August 1, Announcing the Secure Hash Standard.

[41] Odlyzko, A. *Public Key Cryptography*. AT&T Bell Laboratories. New Jersey. S/f.

[42] Pastor, J., Sarasa, M. Criptografía digital. Fundamentos y aplicaciones. Prensas Universitarias de Zaragoza. Zaragoza, España. 1998.

[43] Pointcheval, D. *Le chiffrement asymétrique et la sécurité prouvée*. Habilitation à diriger des recherches. Laboratoire d'Informatique. École Normale Supérieure. Université Paris VII. Paris, France. Juin 2002.

[44] Poss, J. *Introduction à la cryptographie*. École Nationale d'Arts et Métiers. Aix-en-Provence. Juin 2003.

[45] Preneel, B. *Hash Functions*. Version 4. Katholieke Universiteit Leuven. Leuven. May 15, 2004.

[46] Real Academia Española. Diccionario de la lengua española. Real Academia Española. Madrid. 1992. 21ª Edición.

[47] Rivest, R. *Testing Implementations of DES*. MIT Laboratory for Computer Science. Cambridge, Massachussets. February 1985.

[48] Rivest, R., Shamir, A., Adleman, L. *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.* Communications of the ACM. February 1978.

[49] Sale, Tony (Translator). The Bentchley Park translated Enigma Instruction Manual. Traducido en 2001 del documento Der Schlüssel M Verfahren M Allgemein del Comando Supremo de la Marina Alemana, elaborado en 1940. Descargado de http://www.codesandciphers.org.uk/documents/officer/officer1.pdf el 8 de diciembre de 2003.

[50] Salenave Gonçalves, L., Gadis Ribeiro, V. *Um estudo comparativo entre algoritmos de criptografía DES – Lucifer (1977) e AES – Rijndael (2000)*. Universidade Luterana do Brasil (ULBRA)-Centro Universitario Lasalle (UNILASALLE). S/f.

[51] Schneier, B. *A Self-Study Course in Block-Cipher Cryptanalysis*. Counterpane. San José. S/f.

[52] Schneier, B. Applied Cryptography. John Wiley & Sons Inc. 1995.

[53] Schneier, B. *Cryptography, Security and the Future*. Communications of ACM. Vol. 40, No. 1, January 1997. pp. 138. Descargado de http://www.schneier.com/essay-csf.html el 10 de diciembre de 2003.

[54] Schneier, B. *Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)*. Fast Software Encryption. Cambridge Security Workshop Proceedings (December 1993). Springer-Verlag, 1994, Pp. 191-204.

[55] Schneier, B., Whiting, D. A Performance Comparision of the Five AES Finalists. Counterpane Internet Security-Hi/Fn. 7 April 2000

[56] Shirriff, K. *Differential Cryptanalysis of Madryga*. Sun Microsystems Labs. Draft Version. October 19, 1995.

[57] Shannon, C.E. *The Mathematical Theory of Communication.* Reprinted with corrections from The Bell System Technical Journal. Vol. 27. , pp. 379-423, 623-656. July-October, 1948.

[58] Shannon, C.E. *Communication Theory of Secrecy Systems.* The Bell System Technical Journal. 1949. pp. 656-715. 1949.

[59] Simmons, G. (Editor). Contemporary Cryptology. The Science of Information Integrity. IEEE Press. New York. 1992.

[60] Singh, S. The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography. Anchor Books-Knopf Publishing Group. New York. 2000.

[61] Standaert, F., Piret, G., Quisquater, J. *Cryptanalysis of Block Ciphers: A Survey*. Université Catholique de Louvain. UCL Crypto Group Technical Report Series. Technical Report CG-2003/2. Louvain-la-Neuve, 2003.

[62] Stern, J. La science du secret. Éditions Odile Jacob. Paris. 1998.

[63] Stinson, D. *Some Observations on the Theory of Cryprographic Hash Functions*. University of Waterloo. School of Computer Science. Waterloo, Ontario. January 15, 2004.

[64] Vauldenay, S. *On Measuring Resistance to Linear Cryptanalysis.* École Polytechnique Fédérale de Lausanne (EPFL). Lausanne. S/f.

[65] Vauldenay, S. *On the Weak Keys of Blowfish*. École Normale Supérieure – DMI. Laboratoire d'Informatique. Paris. S/f.

[66] Wehenkel, L. Théorie de l'information et du codage. Faculté des Sciences Appliquées. Université de Liège. Liège. 2001.

[67] Welsenbach, M. Cryptography in C and C++. APress. New York, 2001.

[68] Zaccagnini, A. Introduzione alla crittografia. Università degli Studi di Parma. Parma. 2002.

[69] Zimmermann, P. Cryptographie et réseau. Pour la Science. No. 260. Juin 1999. pp. 38-44.

[70] Zimuel, E. Introduzione a la crittografia. Mensa Italia. Francavilla al Mare. Luglio 2002.

ANEXOS

Criptoanálisis

El Criptoanálisis consiste en comprometer la seguridad de un sistema de cifrado. El criptoanálisis y la criptografía son complementarias, ésta última trata principalmente de crear y analizar sistemas de cifrado seguros mientras que la primera, trata de romperlos, demostrando su vulnerabilidad.

No se considera Criptoanálisis el descubrimiento de un algoritmo de cifrado secreto, los algoritmos siempre son conocidos.

En general el Criptoanálisis se suele llevar a cabo estudiando grandes cantidades de pares texto claro-texto cifrado generados con la misma clave. El mecanismo que se emplee para obtenerlos es indiferente, y puede ser resultado de *escuchar* un canal de comunicación, o de la posibilidad de que el objeto que esta siendo atacado *responda* con un texto cifrado cuando se le envía un mensaje. Cuanto mayor sea la cantidad de pares, más probabilidades de éxito tendrá el Criptoanálisis.

La mayoría de estos ataques dirigidos a sistemas de cifrado utilizan métodos probabilísticos o estadísticos que explotan una debilidad estructural, presentada intencional o no intencionalmente (puerta trasera), en el algoritmo de cifrado. A fin de determinar sí dichos ataques son posibles, es necesario examinar la estructura del algoritmo y sus propiedades estadísticas.

A continuación algunos tipos de ataques más conocidos:

Texto claro conocido

Este tipo de ataque se produces cuando el ataque cumple con todas las condiciones siguientes:

1. El criptoanalista tiene acceso completo al algoritmo de cifrado
2. El criptoanalista tiene una cantidad considerable de texto cifrado
3. El criptoanalista conoce el texto en claro de parte de ese texto cifrado
   Se asume el Principio de Kerckhoffs, que establece que la seguridad del cifrado ha de residir exclusivamente en el secreto de la clave, y no en el mecanismo de cifrado.

Fuerza bruta o ataque exhaustivo

Consiste simplemente en probar todas y cada una de las posibles claves del espacio de claves hasta encontrar la correcta.

Texto claro escogido

Ocurre cuando el atacante puede obtener el texto claro correspondiente a textos cifrados de su elección. El atacante cifra una cantidad indeterminada de texto claro y compara el resultado con el texto cifrado original. Este es el caso de los ataques contra el sistema de verificación de usuarios utilizado por Unix, donde un intruso consigue la lista de contraseñas y realiza cifrados de textos en claro de su elección para luego comparar los resultados con las llaves cifradas. A este ataque también se le llama *de diccionario*, debido a que el atacante suele utilizar un archivo llamado `diccionario' con los textos en claro que va a utilizar.

Criptoanálisis diferencial

El criptoanálisis diferencial es un método de ataque basado en un conjunto de pares de texto en claro-texto cifrado que se emplean para determinar el valor de los bits de la llave principal. La

información sobre la llave se obtiene en base a deducciones procedentes de bloques de texto cifrado sobre bloques de texto claro con una diferencia de bits específica entre ellos.

Criptoanálisis lineal

Su funcionamiento se basa en tomar algunos bits del texto claro y efectuar una operación XOR entre ellos, tomar algunos del texto cifrado y aplicarles la misma operación, y análogamente hacer un XOR de los dos resultados anteriores, obteniendo un único bit. Efectuando esa operación a una gran cantidad de pares de texto plano y textos cifrados diferentes se puede observar si se obtienen más ceros o más unos.

Existen combinaciones de bits que, bien escogidas, dan un rango significativo a la medida anteriormente definida, es decir, que el número de ceros (o unos) es apreciablemente superior. Esta propiedad permite poder asignar mayor probabilidad a unas claves sobre otras y de esta forma descubrir la clave que se busca.

Shortcut Attacks

En este tipo de ataque, el adversario explota algunas de las propiedades del algoritmo de cifrado que le permita determinar la clave o el texto claro en mucho menos tiempo que realizando una búsqueda exhaustiva.

Ataques por análisis temporal (Timing Attacks)

Este tipo de ataques se basan en el hecho de que una computadora tarda más tiempo en procesar una multiplicación matemática si los coeficientes de dicha operación son distintos de cero que si alguno de ellos es cero, por lo que un criptoanalista puede obtener información sobre la llave al realizar un estudio sobre el tiempo que emplea el algoritmo en procesar cada unos de los bits de dicha llave. El éxito de un ataque por análisis temporal reside en las multiplicaciones matemáticas.

Cualquier algoritmo de cifrado, para ser considerado seguro, ha de soportar todos estos ataques y otros no citados; sin embargo, el sistema más robusto caerá fácilmente si el emisor o al receptor acceden, forzados o voluntariamente, a revelar información que se supone debe ser secreta.

Ejemplo de la primera iteración de un texto al que se aplica el algoritmo DES

Texto a cifrar: un su cipotío chelito peloemescal
Llave: salarrue

El texto a cifrar, al cual denominaremos M, se representa en formato hexadecimal y binario.

- M en hexadecimal:

u	n		s	u		c	i	p	o	t	i	o		c	h	e	l	i	t	o		p	e	l	u	e	m	e	s	c	a	l					
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33					
55	4E	20	53	55	20	43	49	50	4F	54	49	4F	20	43	48	45	4C	49	54	4F	20	50	45	4C	55	45	4D	45	53	43	41	4C	0D	0A	0	0	0

- M en binario:

Como resultado se obtiene un bloque de 64 bits, el cual se lee de izquierda a derecha, de manera que el primer bit es cero. Seguidamente, el bloque de bits se separa en dos bloques de 32 bits cada uno, denominándolos, mitad izquierda L_0 y la mitad derecha $R_{0:}$

L_0																															
0	1	0	1	0	1	0	1	0	1	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	1	0	1	0	0	1	1
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
R_0																															
0	1	0	1	0	1	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1	1	0	1	0	0	1	0	0	1
33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64

Como ya se ha mencionado, DES opera con bloques de 64 bits y llaves de 56 bits, las cuales son almacenadas como bloques de 64 bits, pero, cada octavo bit es utilizado para revisión de paridad.

La llave

La llave seleccionada es convertida a hexadecimal y binario:

Llave	HEX	Binario							
s	53	0	1	0	1	0	0	1	1
a	41	0	1	0	0	0	0	0	1
l	4C	0	1	0	0	1	1	0	0
a	41	0	1	0	0	0	0	0	1
r	52	0	1	0	1	0	0	1	0
r	52	0	1	0	1	0	0	1	0
u	55	0	1	0	1	0	1	0	1
e	45	0	1	0	0	0	1	0	1

Paso 1: Creación de 16 sub-llaves, cada una con 48 bits de longitud, la cual es permutada de acuerdo a la siguiente tabla:

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

0	1	0	1	0	0	1	1
1	2	3	4	5	6	7	8
0	1	0	0	0	0	0	1
9	10	11	12	13	14	15	16
0	1	0	0	1	1	0	0
17	18	19	20	21	22	23	24
0	1	0	0	0	0	0	1
25	26	27	28	29	30	31	32
0	1	0	1	0	0	1	0
33	34	35	36	37	38	39	40
0	1	0	1	0	0	1	0
41	42	43	44	45	46	47	48
0	1	0	1	0	1	0	1
49	50	51	52	53	54	55	56
0	1	0	0	0	1	0	1
57	58	59	60	61	62	63	64

La llave de 64 bits es permutada, de acuerdo a la tabla que aparee arriba a la izquierda; en ella, el número 57 significa que el bit en la posición 57 de la llave original será el primer bit de la llave permutada K+, seguido por el bit en la posición número 49, de manera que el bit ubicado en la posición 4 de la llave original, será el último bit de la llave permutada.

La llave permutada K+ es:

0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
0	0	1	1	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1
29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56

Ahora, la llave K+ es dividida en mitades, denominando C_0 a la mitad izquierda, y D_0 a la mitad derecha.

Con los valores de C_0 y D_0 definidos, se crean 16 bloques que generarán las 16 sub-llaves que se utilizarán en cada ronda, para ello se utiliza la siguiente tabla que indica el número de vuelta y desplazamiento, a la izquierda, que se realizará tomando como base el valor de C_0 y D_0, donde cada valor generado dependerá de su predecesor, es decir, C_1 D_1, dependerá de C_0 D_0, C_2 D_2 dependerá de C_1 D_1 y así sucesivamente.

C0																											
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1
D0																											
0	0	1	1	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1

Vuelta	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bits desplazados	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Tabla 7 Número de bits de C0 y D0 a desplazar en cada vuelta

Los valores generados son:

C0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1
D0	0	0	1	1	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1
C1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0
C1	0	1	1	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0
C2	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0
D2	1	1	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
C3	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
D3	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	1
C4	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
D4	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	1	0	0
C5	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
D5	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1
C6	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	1
D6	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1
C7	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1
D7	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	0	0
C8	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1
D8	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	0	0	0	1
C9	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
D9	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	0	0	0	1	0
C10	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0
D10	0	0	0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	0	0	0	1	0	0	0
C11	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0
D11	0	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0
C12	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0
D12	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0
C13	0	0	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
D13	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0
C14	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
D14	0	0	1	0	0	1	1	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0
C15	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1
D15	1	0	0	1	1	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0
C16	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1
D16	0	0	1	1	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1

Se procederá a realizar la operación compresión permutación, la cual generará las 16 sub-llaves que serán utilizadas, junto a los bloques de texto, en cada una de las 16 vueltas del algoritmo. La siguiente tabla indica el orden de los bits que serán utilizados para generar las sub-llaves:

14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

Para generar la primera sub-llave, k_1, se utilizan los valores de C_1D_1 antes calculados, por ejemplo:

C1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
D1	0	1	1	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0
	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56

k_1

1	0	1	0	0	0	0	0	1	0	0	1	0	0	1	0	1	1	0	0	1	0	1	0
14	17	11	24	1	5	3	28	15	6	21	10	23	19	12	4	26	8	16	7	27	20	13	2

Para generar la segunda sub-llave, k_1, se utilizan los valores de C_2D_2, así

1	0	1	1	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
41	52	31	37	47	55	30	40	51	45	33	48	44	49	39	56	34	53	46	42	50	36	29	32

k_2

C2	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
D2	1	1	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56

1	0	1	1	0	0	0	0	0	0	0	1	0	0	1	0	1	1	0	1	0	0	1	0
14	17	11	24	1	5	3	28	15	6	21	10	23	19	12	4	26	8	16	7	27	20	13	2

0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	0	0	1	1	0
41	52	31	37	47	55	30	40	51	45	33	48	44	49	39	56	34	53	46	42	50	36	29	32

Al realizar el proceso anterior con cada uno de los valores de $C_1D_{1,...,}$ $C_{16}D_{16,}$ se obtienen las 16 sub-llaves que serán utilizadas en cada una de las 16 vueltas del algoritmo.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
K1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	1	0	1	1	0	0	1	0	1	0	1	0	1	1	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
k2	1	0	1	1	0	0	0	0	0	0	0	1	0	0	1	0	1	1	0	1	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	0	0	1	1	0
k3	0	0	1	1	0	1	0	0	0	1	0	1	1	0	1	0	0	1	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	1	1	0	1	0	0	0	0	1
K4	0	0	0	0	0	1	1	0	0	1	1	1	0	0	0	1	0	1	0	1	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	1	1	0	1
K5	0	0	0	0	1	1	1	0	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	0	0	1	0	1	0	0	1	0	0	0	1	1	0	0	1	0	0	1	0
K6	0	1	0	0	1	1	1	0	1	0	0	0	0	0	0	1	0	0	1	0	1	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	1	0	0	1	0	1	0	0	1
K7	1	0	0	0	1	0	1	1	1	0	0	0	0	0	0	1	1	0	1	0	1	0	0	1	0	1	0	0	1	0	1	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0
K8	1	0	0	1	1	0	0	1	0	0	0	0	1	0	1	0	1	0	0	0	1	0	1	1	0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	1	1	1	1	0	0
K9	0	0	1	1	1	0	0	1	0	0	0	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	0	0	0	1	0	0	1	1	0	0	1	1	0	0	0	0	0	0	1	0	0	0
K10	0	0	1	1	0	0	0	0	0	0	1	1	1	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	1	1	0	1	1	0	0	1	0	1	0	0	0	0
K11	0	0	0	1	0	0	0	0	0	0	1	0	1	1	0	0	0	1	0	1	0	1	0	0	1	0	1	1	1	0	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
K12	0	1	0	0	0	1	0	0	0	1	1	0	1	1	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0	0	0	0	0	0	1	0	0
K13	1	1	0	0	0	1	1	0	1	0	1	0	0	1	0	1	0	0	1	0	0	1	0	0	1	0	0	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0
K14	1	1	0	0	1	0	1	0	1	0	0	0	0	1	1	0	0	0	1	0	0	0	1	1	1	0	1	0	0	0	0	1	0	1	0	0	0	0	1	0	1	1	0	0	0	0	0	0
K15	1	1	1	0	1	0	0	1	1	0	0	1	0	0	1	0	0	0	1	0	1	0	1	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	1	1
K16	1	0	1	0	0	0	0	1	1	0	0	1	0	0	1	0	1	0	1	0	1	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	1	0	1

Luego de generadas las 16 sub-llaves se inicia el procesamiento de los bloques de texto claro.

Paso 2: Cifrado de los bloques de 64 bits de datos

Se ejecutará la función permutación expansión para luego realizar la permutación inicial (PI) de todos los bits. Se reordenarán los bits de acuerdo al orden especificado en la tabla que abajo se presenta.

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

IP

L_0																															
0	1	0	1	0	1	0	1	0	1	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	1	0	1	0	0	1	1
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
R_0																															
0	1	0	1	0	1	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1	1	0	1	0	0	1	0	0	1
33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64

0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0	1	0	0	1	0	0	1	0	1	0
57	49	41	33	25	17	9	1	59	51	43	35	27	19	11	3	61	53	45	37	29	21	13	5	63	55	47	39	31	23	15	7

1	1	0	1	1	0	1	1	0	0	0	1	1	0	0	1	0	0	0	1	0	0	1	1	1	1	0	1	1	0	0	1
58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4	62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8

El resultado obtenido anteriormente se divide en mitades, asignándoles R_0 a la mitad izquierda y L_0 a la mitad derecha:

R0	1	1	0	1	1	0	1	1	0	0	0	1	1	0	0	1	0	0	0	1	0	0	1	1	1	1	0	1	1	0	0	1
L0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0	1	0	0	1	0	0	1	0	1	0

Ahora se inician las 16 iteraciones del algoritmo, utilizando la función F, que operará los bloques de 32 bits y las sub-llaves de 48 bits, para producir un bloque de 32 bits.

Como resultado de la última vuelta se obtendrá el valor $L_{16}R_{16}$. Eso quiere decir que en cada iteración, se tomarán los 32 bits del resultado anterior, para tomarlos como los 32 bits de la siguiente vuelta.

Se utilizará la siguiente fórmula:
$L_n = R_{n-1}$
$R_n = L_{n-1} + f(R_{n-1}, K_n)$
De manera que para la primera vuelta, n = 1:

$L_1 = R_0$

$R_0 = L_0 + f\ (R_0, K_1)$

K_1 101000001001001011001010101101100000010000000100

L_1 =110110110001100100   01001111011001

$R_1 = L_0 + f(R_0, K_1)$

Para calcular f, se expande el bloque R_0 , partiendo de 32 bits, se generarán 48, utilizando la tabla de expansión, E, abajo presentada:

32	1	2	3	4	5	4	5
6	7	8	9	8	9	10	11
12	13	12	13	14	15	16	17
16	17	18	19	20	21	20	21
28	29	28	29	30	31	32	1

Al resultado generado por la tabla anterior, se le llama $E(R_0)$. Los bits ordenados de R_0 son:

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

R0	1	1	0	1	1	0	1	1	0	0	0	1	1	0	0	1	0	0	0	1	0	0	1	1
E(R0)	1	1	1	0	1	1	1	1	0	1	1	0	1	0	0	0	1	1	1	1	0	0	1	0
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

R0	1	1	1	0	1	1	0	0	1															
E(R0)	1	0	0	0	1	0	1	0	0	1	1	1	1	1	1	0	1	1	1	1	0	0	1	1
	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48

Es importante observar, que el bloque original de 4 bits ha sido expandido a un bloque de 16 bits.

R0																														
1	1	0	1	1	0	1	1	0	0	0	1	1	0	0	1	0	0	0	1	0	0	1	1	1	1	0	1	1	0	0	1
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

Ahora, se aplicará la función XOR a la salida anterior R0 con las sub-llaves anteriormente creadas, K_n XOR E(R_{n-1}).

Para el caso actual K_1 XOR E(R_0):
K1 = 10100000100100101100101010110110000001000000010 0
$E(R_0)$ = 11101111011010001111001010001010011111101111 0011
K_1 XOR $E(R_0)$ = 011000 010001 011110 111010 100001 100110 010100 100111

K1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	1	0	1	1	0	0	1	0	1	0	1	0	1	1	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0
E(R0)	1	1	1	0	1	1	1	1	0	1	1	0	1	0	0	0	1	1	1	1	0	0	1	0	1	0	0	0	1	0	1	0	0	1	1	1	1	1	1	0	1	1	1	1	0	0	1	1
XOR	0	1	0	0	1	1	1	1	1	1	1	1	1	0	1	0	0	0	1	1	1	0	0	0	0	0	1	1	1	1	0	0	0	1	1	1	1	0	1	1	1	1	1	1	0	1	1	1

Aún no se finaliza con las operaciones de la función F. Hasta este punto, se ha expandido R0 de 32 a 48 bits, utilizando la tabla de extensión, se ha aplicado XOR a dicho resultado con la llave. El resultado obtenido se muestra en la tabla siguiente, en grupos de seis bits.

B1	0	1	0	0	1	1
B2	1	1	1	1	1	1
B3	1	0	1	0	0	0
B4	1	1	1	0	0	0
B5	0	0	1	1	1	1
B6	0	0	0	1	1	1
B7	1	0	1	1	1	1
B8	1	1	0	1	1	1

A continuación se hará uso de las tablas de sustitución. Cada grupo de seis bits de la tabla anterior proporcionará una dirección en una caja de sustitución diferente. De manera que, los ocho grupos de seis bits con los que actualmente contamos, serán transformados en ocho grupos de cuatro bits, es decir, 32 bits.

La fórmula a utilizar será: $S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)$, donde Sn son las cajas de sustitución. Donde $S_i(B_i)$ se refiere a la salida de las cajas de sustitución.

Cada caja de sustitución toma bloques de 6 bits como entrada y genera bloques de 4 bits de salida. El proceso se describe a continuación:

Sea S_1 la primera caja de sustitución y B_1 el primer bloque de seis bits, entonces $S_1(B)$ se determina:

- El primer y último bit de B representan, en base 2, un número decimal, en un rango de cero a tres, o de 00 a 11 en binario, a este número se le denomina *i*.

- Los cuatro bits restantes, de B, representan, en base 2, un número decimal, de cero a quince, a este número se le denomina *j*.

En las tablas de sustitución se busca el valor dado por el par ordenado (i,j), el cual será la salida, en decimal, generada por cada tabla, que posteriormente será convertida a binario.

							Primer Bit	Último Bit	Número de fila	Bits Restantes				Número de Columna
B1	0	1	0	0	1	1	0	1	1	1	0	0	1	9
B2	1	1	1	1	1	1	1	1	3	1	1	1	1	15
B3	1	0	1	0	0	0	1	0	2	0	1	0	0	4
B4	1	1	1	0	0	0	1	0	2	1	1	0	0	12
B5	0	0	1	1	1	1	0	1	1	0	1	1	1	7
B6	0	0	0	1	1	1	0	1	1	0	0	1	1	3
B7	1	0	1	1	1	1	1	1	3	0	1	1	1	7
B8	1	1	0	1	1	1	1	1	3	1	0	1	1	11

En la tabla anterior podemos observar los valores (i,j) que se utilizarán para cada una de las cajas de sustitución.

S1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

(i,j) = (1,9) = 10 = 1010

S2

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

(i,j) = (3,15) = 15 = 1111

S3

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

(i,j) = (2,4) = 3 = 0011

S4

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

(i,j) = (2,12) = 1 = 0001

S5

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6

4 2 1 11 10 13 7 8 15 9 12 5 6 3 0 14
11 8 12 7 1 14 2 13 6 15 0 9 10 4 5 3

(i,j) = (1,7) = 6 = 0110

S6

12 1 10 15 9 2 6 8 0 13 3 4 14 7 5 11
10 15 4 2 7 12 9 5 6 1 13 14 0 11 3 8
9 14 15 5 2 8 12 3 7 0 4 10 1 13 11 6
4 3 2 12 9 5 15 10 11 14 1 7 6 0 8 13

(i,j) = (1,3) = 15 = 1111

S7

4 11 2 14 15 0 8 13 3 12 9 7 5 10 6 1
13 0 11 7 4 9 1 10 14 3 5 12 2 15 8 6
1 4 11 13 12 3 7 14 10 15 6 8 0 5 9 2
6 11 13 8 1 4 10 7 9 5 0 15 14 2 3 12

(i,j) = (3,7) = 14 = 1110

S8

13 2 8 4 6 15 11 1 10 9 3 14 5 0 12 7
1 15 13 8 10 3 7 4 12 5 6 11 0 14 9 2
7 11 4 1 9 12 14 2 0 6 10 13 15 3 5 8
2 1 14 7 4 10 8 13 15 12 9 0 3 5 6 11

(i,j) = (3,7) = 14 = 1110

Para la primera vuelta, se obtuvieron los siguientes resultados, de las cajas de sustitución:

$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)$ =
1010 1111 0011 0001 0110 1111 1110 1110

Para la última etapa de la función F es hacer la permutación P del resultado generado por las cajas de sustitución, para así obtener el valor F. La fórmula a utilizar es la siguiente:

$f = P(S_1(B_1)S_2(B_2)...S_8(B_8))$.

La permutación P genera 32 bits de salida de una entrada de también 32 bits. La siguiente tabla define la permutación P:

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	31	10
32	27	3	9
19	13	30	6
22	11	4	25

S1(B1)S2(B2)S3(B3)S4(B4)S5(B5)S6(B6)S7(B7)S8(B8)																															
1	0	1	0	1	1	1	1	0	0	1	1	0	0	0	1	1	0	1	1	1	1	1	1	0	1	1	1	1	1	0	1
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

Luego de realizada la permutación, se obtiene:

1	1	1	1	1	1	1	1	1	0	1	1	1	0	0	0	0	1	0	0	1	1	1	0	1	0	1	1	1	1	0	0
16	7	20	21	29	12	28	17	1	15	23	26	5	18	31	10	2	8	31	10	32	27	3	9	19	13	30	6	22	11	4	25

F = 11111111101110000100111010111100

$R_1 = L_0 + f(R_0 , K_1)$

$R_{1 =}$
01010101010011100010000001010011
11111111101110000100111010111100

10101010111001100100111011111111

$R_{1 =}$10101010111001100100111011111111

En la siguiente vuelta, se utilizará $L_2 = R_1$, el cual es el bloque que se acaba de obtener, y entonces, se deberá calcular $R_2 = L_1 + f(R_1, K_2)$, y así sucesivamente para el resto de vueltas. Al final de las 16 vueltas se tendrán los bloques L_{16} y R_{16}. A los cuales se les invierte el orden, para obtener $R_{16}L_{16}$ y aplicar la permutación final, tal como indica la siguiente tabla:

IP^{-1}

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Para el proceso de descifrado, se deberá hacer el proceso anterior, invirtiendo el orden de las sub-llaves aplicadas.

Tabla comparativa entre cifrado de llave secreta y cifrado de llave pública

	Llave secreta	Llave pública
Gestión de claves	El número de llaves a memorizar es muy elevado	Solo es necesario memorizar la llave privada del emisor
Longitud y universo de llaves[81]	La llave suele ser del orden de las centenas de bits	La llave suele ser del orden de los miles de bits
Vida de las llaves[82]	Muy corta. Normalmente se usa una por sesión, lo que implica una vida de segundos o minutos	Larga. Se aprovecha que es entregada y gestionada por un tercero
Autenticación[83]	Se puede autenticar al mensaje, pero no al emisor de forma sencilla y clara	Al haber una clave pública y otra privada, es factible la autenticación del mensaje y el emisor
Velocidad[84]	Muy alta. Es el algoritmo que cifra el mensaje. Normalmente del orden de cientos de Mbytes/s cuando se realiza en hardware	Muy baja. Se usa para el intercambio de llave y la firma digital. Normalmente es del orden de cientos de Kbytes/s cuando se realiza en hadrware
Usos	La velocidad de cifra es muy alta y por ello se usa para realizar la función de cifra de la información. Además, con claves de sólo unas centenas de bits se obtiene una seguridad razonablemente alta pues su no linealidad y algoritmo hace que en condiciones convencionales el único ataque que puede prosperar sea el de la fuerza bruta.	Cuando usando la clave pública del destino se hace el intercambio de llaves de sesión de una cifra con sistemas simétricos (decenas a centenas de bits), o cuando al usar la clave privada de origen, se firma digitalmente un resumen (decenas a centenas de bits) del mensaje obtenido con una función hash.

[81] En cuanto al espacio de claves, no son comparables los sistemas simétricos con los asimétricos. Para atacar un sistema asimétrico no se buscará en todo el espacio de llaves como debería hacerse en los sistemas simétricos.
[82] En cuanto a la vida de una llave, en los sistemas simétricos ésta es muchísimo menor que la de las usadas en los asimétricos. La llave de sesión es aleatoria, en cambio la asimétrica es propia del usuario.
[83] Los sistemas simétricos tienen una autenticación más pesada y con una tercera parte de confianza. Los asimétricos permiten una firma digital verdadera, eficiente y sencilla.
[L] los sistemas simétricos son de cien a mil veces más rápidos que los asimétricos. En software la velocidad de cifra es más baja.

Tabla comparativa de estructuras de algunos algoritmos de **cifrado de bloques**

Algoritmo	Bloque de datos (bits)	Longitud de llave (bits)	Vueltas
Lucifer	128	128	16
DES	64	56	16
IDEA	64	128	8
Madryga[85]	64	64	
Blowfish	64	Variable	16
3-DES	64	168	16x3
Skipjack	64	80	32
Rijndael	128	128 o más	Flexible

[85] Las dimensiones del bloque de datos y la llave de Madryga pueden variar, pero desde su lanzamiento se recomendó estandarizarlas a 64 bits con el fin de buscar cierta compatibilidad con el DES. En cuanto a las vueltas, su proceso de cifrado consta de dos ciclos anidados, de los cuales el externo consta de 8 iteraciones del interno, que a su vez consta de 8 operaciones sobre el bloque de datos.

Tabla comparativa del rendimiento de algunos algoritmos de cifrado de bloque

Algoritmo	Velocidad[86]	Requerimientos de implementación[87]	Vulnerabilidad
Lucifer	Media	Razonables	Media
DES	Media (35Kb/s)	Razonables	Media
IDEA	Media (53Kb/s)	Razonables	Media
Madryga	Media-alta	Bajos	Alta
Blowfish	Alta (110-182 Kb/s)	Bajos	Baja
3-DES	Baja (12 Kb/s)	Altos	Baja
Skipjack[88]	Media-alta	Bajos	Baja
Rijndael	Alta	Razonables	Baja

[86] En los casos en los que se indica un número, el valor ha sido tomado de Pastor, J., Sarasa, M. Criptografía digital. Fundamentos y aplicaciones. Prensas Universitarias de Zaragoza. Zaragoza, España. 1998. Según refieren estos autores, las pruebas que arrojaron los valores que se indican fueron efectuadas en computadoras con procesadores 486. Como es de entender, las velocidades de procesamiento de los microprocesadores con los que actualmente cuentan las computadoras, exceden por mucha las de los 486, de manera que los valores que se indican se presentan únicamente para fines de comparación.

[87] Este aspecto se refeire a cuán fácil es implementar en hardware y/o software cada uno de los algoritmos.

[88] Está patentado y su implementación únicamente está autorizada en hardware; hay disponibles implementaciones en circuitos integrados cuyo precio no suele ser elevado.

El Modelo OSI

El modelo fue iniciado en 1974 por IBM, para redes de computadoras, era conocido como SNA (System Network Architectura), pero su versión definitiva surgió en 1984, luego de ser perfeccionado por la ISO (International Standards Organization) entidad que le cambió el nombre a OSI (Open Systems Interconnection).

La finalidad del modelo OSI es permitir la cooperación entre sistemas abiertos. Un sistema abierto es aquel conjunto de computadoras, material lógico, periféricos, etc., que forman un todo autónomo capaz de procesar y/o transferir información.

El modelo OSI está compuesto por 7 capas, entre las cuales se realiza una comunicación vertical denominada *Servicio*. Se realiza también una comunicación horizontal entre distintos sistemas abiertos lo que se denomina *Protocolo.* Cada capa ofrece un servicio a la capa inmediatamente superior y requiere los servicios de la inferior.

Este modelo no es una arquitectura de red en sí mismo, dado que no se especifican en forma exacta los servicios y protocolos que se utilizarán en cada capa, sino que solamente se indica la funcionalidad de cada una de ellas.

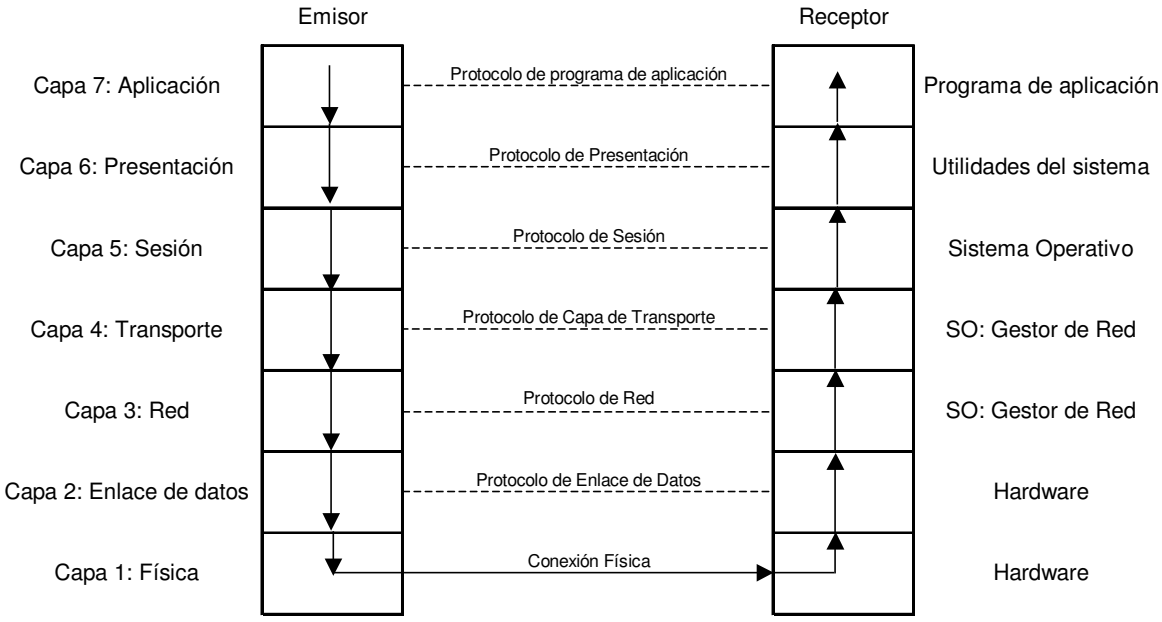


Tabla 1. Capas del modelo OSI

No	Capa	Función	Dispositivos y protocolo
7	Aplicación	Proporciona comunicación entre dos procesos de aplicación.	X.400
6	Presentación	Interpretación de los datos	Compresión, cifrado
5	Sesión	Diálogos de control	Gateway
4	Transporte	Integridad de los mensajes	Gateway. IP, IPX
3	Red	Establece las comunicaciones y determina el camino que tomarán los datos en la red	Router IP, IPX
2	Enlace	Divide el flujo de bits en paquetes, intercambiándolos mediante el uso de protocolos	Puentes (Bridges) HDLC y LLC
1	Físico	Transmisión del flujo de bits a través del medio	Cables, tarjetas, hubs

Tabla 2. Capas, funciones, dispositivos y protocolos del modelo OSI

Los tres niveles inferiores están orientados al acceso del usuario y comunicaciones de datos; el cuarto nivel al transporte extremo a extremo de la información, y los tres superiores a la aplicación.

Capa Física – 1

Esta capa se ocupa de la transmisión de bits a lo largo de un canal de comunicación, y además garantiza que un bit que se manda llegue con el mismo valor que se envió.

En esta capa se encuentran los medios materiales para la comunicación como las placas, cables, conectores, es decir los medios mecánicos y eléctricos.

Capa de Enlace – 2

Es la capa que traslada los mensajes desde y hacia la capa física a la capa de red. Especifica como se organizan los paquetes de datos cuando se transmiten en un medio particular.

Se encarga también de la detección y control de errores ocurridos en la capa física, del control del acceso a dicha capa, la integridad de los datos y la fiabilidad de la transmisión.

Capa de Red – 3

Esta capa se encarga de la transmisión de los paquetes, y de encaminar cada uno en la dirección adecuada. Además obtiene los parámetros de calidad del servicio y notificación de errores.

Capa de Transporte – 4

La capa de transporte se encarga de garantizar la fiabilidad del servicio, describe la calidad y naturaleza del envío de datos. Para ello divide el mensaje recibido de la capa de sesión en paquetes, los numera correlativamente y los entrega a la capa de red para su envío.

Capa de Sesión – 5

Esta capa es una extensión de la capa de transporte que ofrece control de diálogo y sincronización, aunque son pocas las aplicaciones que hacen uso de ella, por ejemplo, las comunicaciones de Internet no la utilizan.

Uno de los servicios de esta capa es la gestión y sincronización de los datos intercambiados entre los usuarios de una sesión.

Capa de Presentación – 6

Ésta permite la representación de la información que las entidades de aplicación comunican o mencionan en su comunicación. Es la responsable de que la información se entregue al proceso de aplicación de manera que pueda ser entendida y utilizada.

Es en esta capa en la cual se hace uso de algoritmos criptográficos.

Capa de Aplicación – 7

La de aplicación es el medio por el cual los procesos de aplicación acceden al entorno OSI. Por ello, ésta capa no interactúa con una superior a ella. La transferencia de archivos es una de las aplicaciones más comunes de este nivel.

Cifrado en la capa de Presentación

El nivel de presentación se ocupa de la sintaxis, es decir la representación de los datos extremo a extremo. Así pues es responsable de alcanzar un acuerdo en los códigos y formatos que se usarán en el intercambio de datos de aplicación durante una sesión, así como también del formateo de datos para su correcta salida a una impresora o a una determinada pantalla.

La capa de presentación trata todos los problemas relacionados con la representación de los datos transmitidos. Por lo tanto incluye los aspectos de: conversión, cifrado y compresión de datos.

El cifrado, no es un elemento que pertenece en exclusiva a la capa de presentación sino que se puede encontrar en otras capas.

Cifrado de Enlace

En este caso el cifrado se realiza en la capa física. Para ello se utiliza una llave en cada computadora participante de la comunicación, y el medio físico, de manera que cada bit que sale de la máquina emisora sufre un proceso de cifrado, y a cada bit que entra en la máquina receptora se le practica el proceso inverso.

Cifrado de Transporte

Si se introduce el cifrado en la capa de transporte se ocasiona que el cifrado se realice en la sesión completa, esto ocasionará una sobrecarga de trabajo de cifrado que en muchas ocasiones será innecesario para algunos de los datos cifrados.

Cifrado de Presentación

Es la mejor solución ya que el cifrado es sufrido sólo por aquellas partes de los datos que sean consideradas necesarias, consiguiendo de este modo que la sobrecarga del proceso de cifrado sea menor.

SSL - Secure Sockets Layer

El protocolo SSL fue desarrollado por Netscape, y su primera versión se hizo pública en 1994, la cual nunca fue implementada. Unos pocos meses después surgió SSL 2.0 una versión mejorada de su antecesora, la cual se implementó a pesar de los múltiples reportes acerca de sus errores de diseño.

En noviembre de 1995 Netscape publicó la especificación para SSL 3.0. El objetivo de Netscape era crear un canal de comunicaciones seguro entre un cliente y un servidor que fuese independiente del sistema operativo usado por ambos y que se beneficiara de forma dinámica y flexible de los nuevos esquemas de cifrado disponibles.

El protocolo puede ser implementado con llaves de sesión de en 40 ó 128 bits. El concepto de llave de sesión se refiere al par de llaves que un servidor y un cliente comparten, las cuales son utilizadas en el proceso de cifrado en cada sesión de SSL. Entre más larga sea la llave de sesión, más difícil será romper el cifrado de la sesión SSL y por lo tanto leer la información transmitida.

SSL fue diseñado como un protocolo seguro de propósito general y no teniendo en mente las necesidades específicas del comercio electrónico.

El protocolo toma ventaja de los algoritmos de llave privada de y de los da llave pública. Con los de llave privada autentica a los clientes y servidores, y con los de llave privada cifra los datos de aplicación.

SL opera entre la capa de transporte y la de sesión del modelo OSI y está formado, a su vez, por dos capas y cuatro protocolos: Registro, Handshake, Alerta y Chage Cipher Spec.

El protocolo de registro (Record Protocol) se encarga de encapsular el trabajo de los elementos de la capa superior, construyendo un canal de comunicaciones entre los dos extremos que participan en la comunicación.

El protocolo Handshake es el encargado de intercambiar la llave que se utilizará para crear un canal seguro mediante un algoritmo eficiente de cifrado simétrico. También es responsabilidad de este protocolo coordinar los estados de ambos extremos de la transmisión.

El protocolo de Alerta es el encargado de señalizar problemas y errores concernientes a la sesión SSL establecida.

El protocolo Change Cipher Spec está formado por un único mensaje consistente en un único byte de valor 1 y se utiliza para notificar un cambio en la estrategia de cifrado.

SSL trabaja de la siguiente forma:

1. Se intercambia una clave mediante un algoritmo de cifrado asimétrico. Mediante esa clave se establece un canal seguro utilizando para ello un algoritmo simétrico previamente negociado.
2. Se toman los mensajes a ser transmitidos, se fragmentan en bloques, se comprimen, se aplica un algoritmo hash para obtener un resumen que es concatenado a cada uno de los bloques comprimidos para asegurar la integridad de los mismos, se realiza el cifrado y se envían los resultados.

SSL comparado con S-HTTP, PCT, TLS e IPSec

Los protocolos S-HTTP, PCT, TLS e IPSec fueron desarrollados, al igual que SSL, como protocolos de carácter general para asegurar la información transmitida a través de un canal de comunicaciones.

Todos siguen el mismo esquema: en primer lugar se invoca un mecanismo para, mediante criptografía asimétrica, intercambiar una clave secreta de longitud suficiente y, en segundo lugar, se utiliza esta clave para cifrar la información transmitida mediante un algoritmo simétrico mucho más eficiente.

S-HTTP, Secure http

El protocolo Secure HTTP o S-HTTP fué posiblemente el primer protocolo de seguridad implementado para Internet. Desarrollado por Enterprise Integration Technologies Inc. (EIT) junto con RSA Data, se hizo público en forma de borrador en Junio de 1994.

S-HTTP es una extensión del protocolo HTTP cuya finalidad es la transmisión de datos de forma segura sobre la web entre un cliente y un servidor. Trabaja sobre la capa de aplicación cifrando el contenido de los mensajes entre las dos máquinas usando para ello un sistema de cifrado basado en un par de llaves pública y privada.

S-HTTP es un protocolo muy flexible que permite que las opciones de firma y cifrado se usen de forma opcional.

La principal diferencia, en cuanto a diseño, con SSL es que S-HTTP fue desarrollado para la transmisión de mensajes individuales de forma segura mientras que SSL se diseñó para establecer una conexión segura permanente entre dos computadoras.

Puesto que S-HTTP es un protocolo situado en la capa de aplicación es capaz de proporcionar características de no repudio de mensajes de forma individualizada a través de las firmas digitales, mientras que SSL, un protocolo de bajo nivel que opera en la capa de transporte, no lo hace.

S-HTTP, sin embargo, se definió como una extensión de HTTP y sus servicios sólo están disponibles para este protocolo.

S-HTTP es más flexible que SSL. Cada aplicación puede configurar el nivel de seguridad que necesita permitiendo mantener más conexiones o responder más rápidamente.

PCT, Private Communication Technology

PCT, Private Communication Technology, fue la respuesta de Microsoft a la creación de Netscape. La primera versión beta fue lanzada al mercado en septiembre de 1995.

PCT fue diseñado para soportar transacciones comerciales de forma segura y espontánea y, al igual que SSL, opera en la capa de transporte siendo, independiente de la aplicación que lo maneja. PCT incorpora mecanismos de clave pública/privada de RSA para cifrado y autenticación de ambos extremos de la comunicación.

Las principales diferencias entre PCT y SSL están en la fase de negociación: Uno de los más novedosos mecanismos que aporta PCT es que separa las negociaciones y mecanismos de autenticación y cifrado.

El amplio uso del SSL ha dejado a este nuevo protocolo sin ninguna posibilidad de competencia. En Mayo de 2000 y como último intento, Microsoft anunció las especificaciones para una versión convergente de SSL y PCT que vendría a llamarse STLP (Secure Transport Layer Protocol) de la cual nunca más se ha publicado algo y que no ha llegado a tener ninguna implementación conocida.

Transport Layer Security, TLS

TLS se hizo público en enero de 1999. Se construyó a partir de las especificaciones de SSL 3.0 y a veces se le denomina como SSL 3.1.

Las principales diferencias entre SSL 3.0 y TLS 1.0 son las siguientes:

- En SSL 3.0 si el servidor solicita un certificado al cliente para que se autentique, este debe de responder con él o con un mensaje de alerta advirtiendo de que no lo tiene. En TLS 1.0 si el cliente no posee certificado no responde al servidor de ninguna forma a este requerimiento.

- Cálculo de las claves de sesión. El mecanismo utilizado para construir las claves de sesión es diferente en TLS 1.0.

- TLS utiliza un mecanismo diferente y más seguro en el cálculo de las funciones resumen.

- TLS 1.0 introduce nuevos códigos de alerta no contemplados por SSL 3.0

- TLS 1.0 introduce un nuevo mecanismo en el relleno de los bloques para frustrar ataques basados en el análisis de la longitud de los mensajes.

IPSec

IPSec proporciona autenticación, confidencialidad e integridad de datos, trabaja en la capa de red y, gracias a su posición en el modelo OSI, es capaz de trabajar con UDP y otros protocolos de la capa de transporte, contrariamente a SSL.

Por las razones anteriores algunas personas a IPSec como el verdadero sustituto de SSL mientras que otras insisten en que ambos coexistirán juntos puesto que ofrecen soluciones óptimas para diferentes problemas.

Problemas con Otros Protocolos de la Capa de Transporte

SSL se encuentra entre la capa de transporte y la de sesión y soporta, de forma transparente, todos los protocolos que se sitúan sobre él mismo: HTTP, FTP, IMAP, LDAP, etc.

El protocolo SSL no fue diseñado para trabajar con otros protocolos de la capa de transporte diferentes de TCP y, sobre todo, no orientados a conexión, como UDP, y otros que han caído en desuso como es el caso de IPX.

Aplicación para cifrado y descifrado de datos

Como complemento de la información presentada en el presente trabajo se desarrolló una aplicación de software, la que se denominó Kriptonita, que muestra el proceso de cifrado y descifrado de información, para el desarrollo de los algoritmos de cifrado se utilizó lenguaje C y para la pantalla de captura de datos se utilizó Microsoft Visual Studio.Net

Para la utilización de Kriptonita se deberán seguir los siguientes pasos:

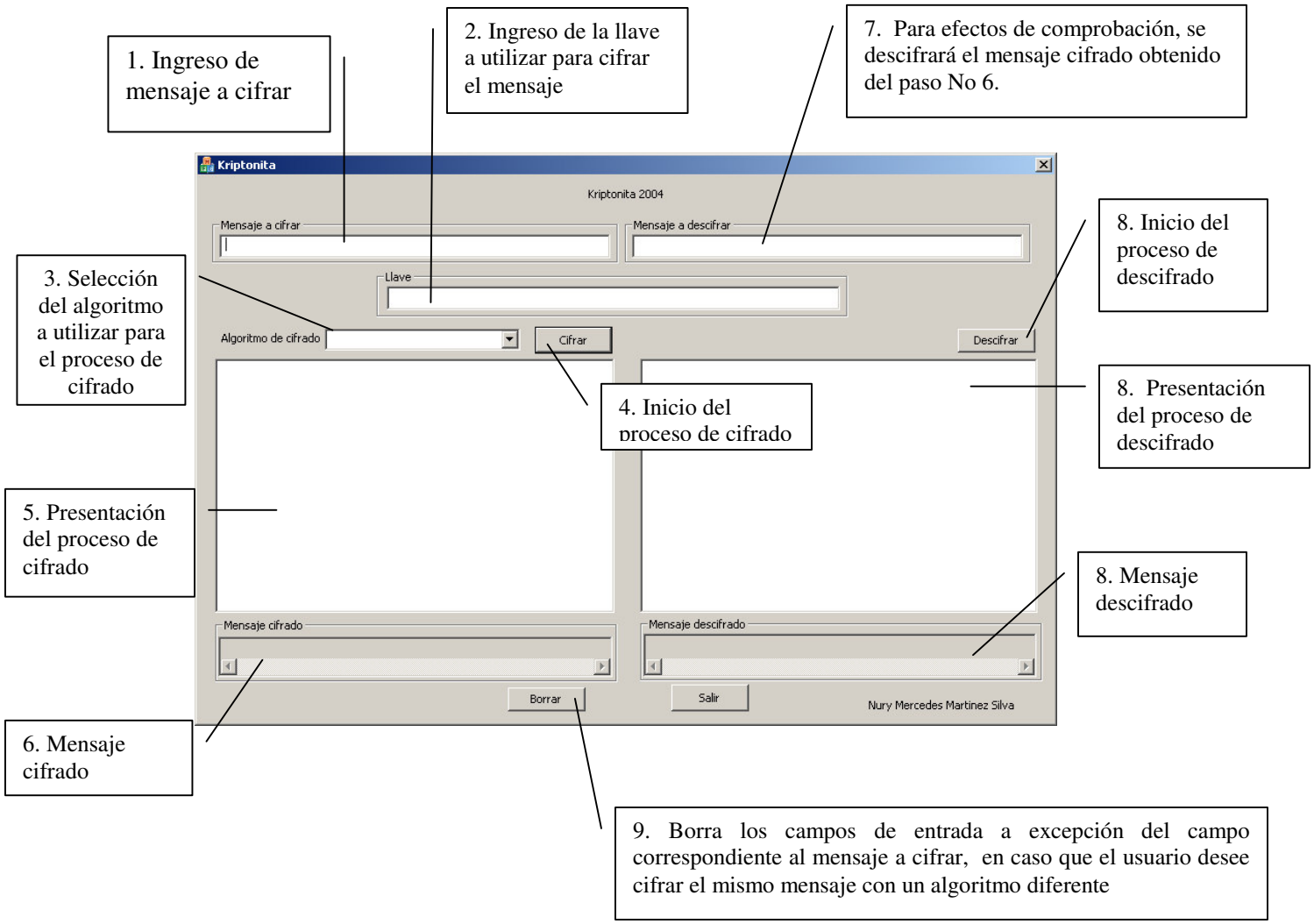


Figura 1 Descripción de la pantalla de ingreso de Kriptonita

Ejemplos de utilización

A continuación algunos ejemplos de utilización de Kriptonita. Es importante mencionar que la llave es complementada con ceros, según la longitud de la llave requerida por el algoritmo seleccionado.

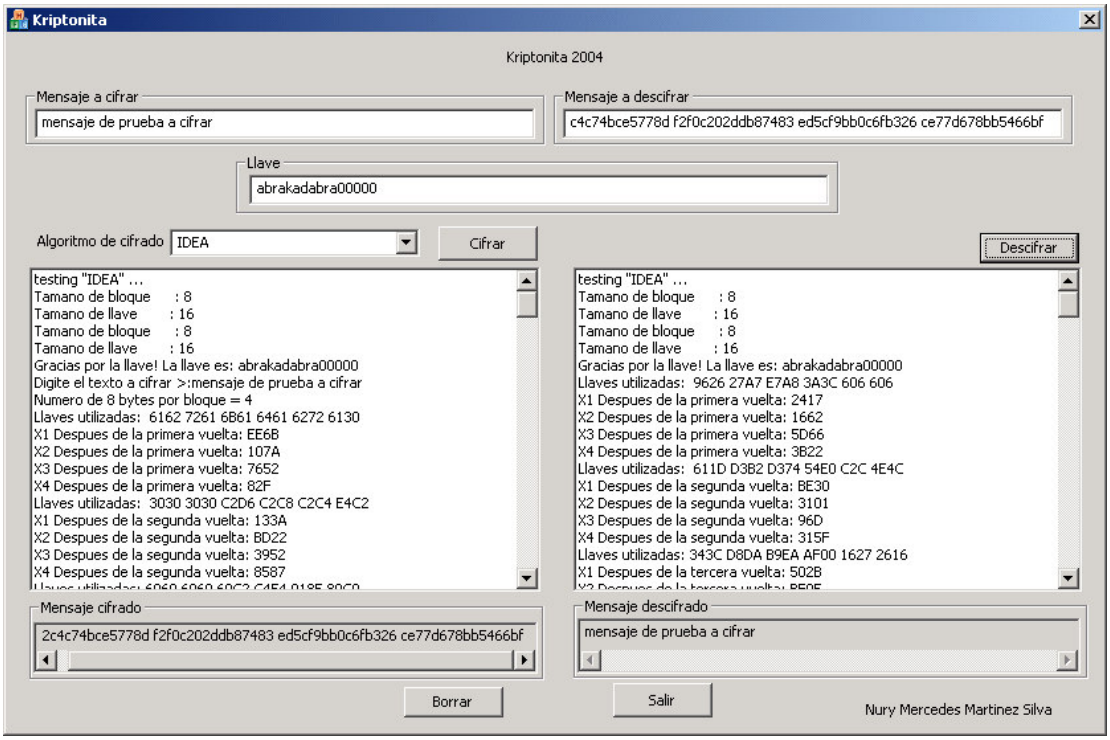


Figura 2 Ejemplo de cifrado utilizando IDEA

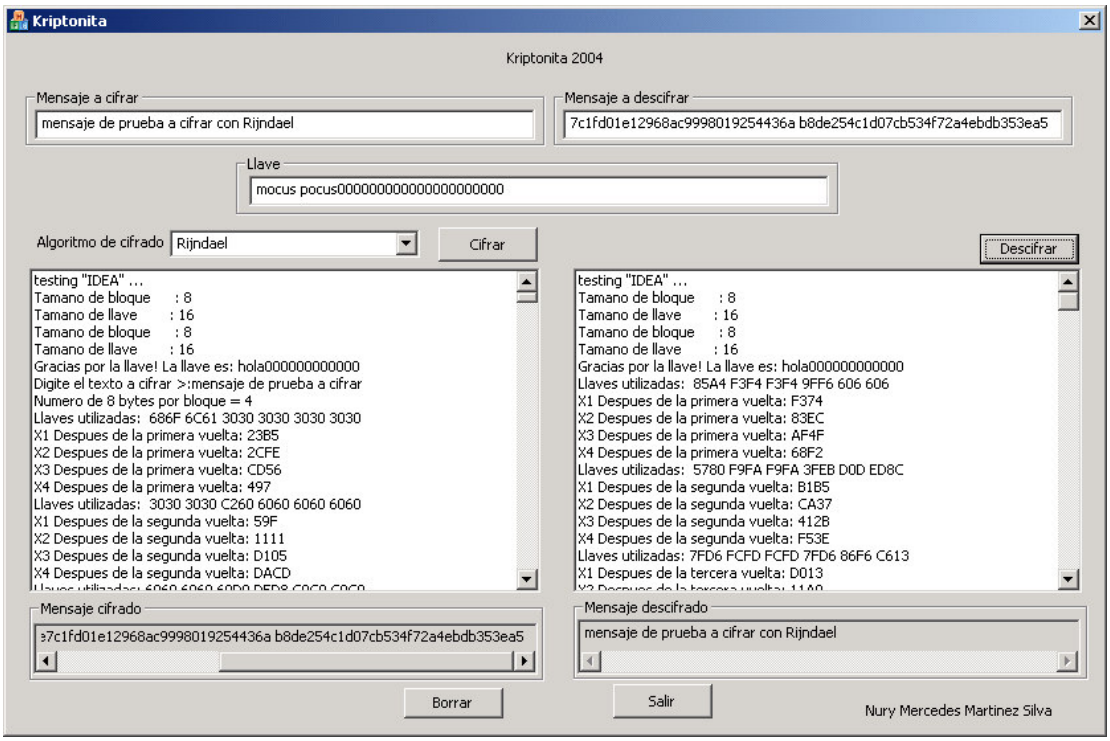


Figura 3 Ejemplo de cifrado utilizando Rijndael

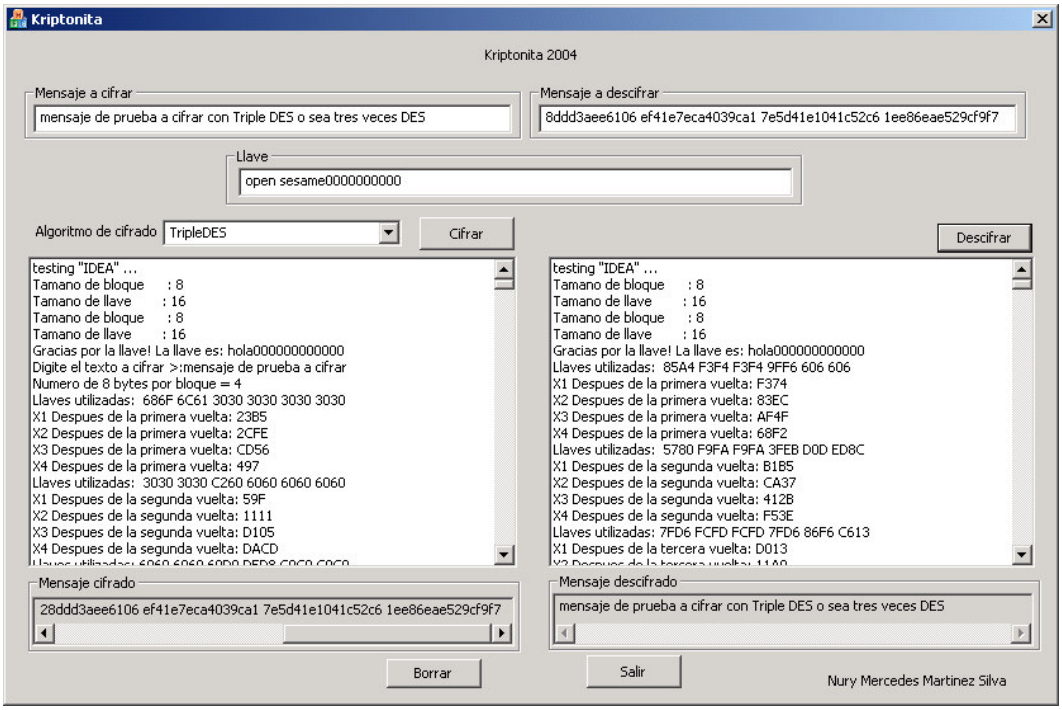


Figura 4 Ejemplo de cifrado utilizando Triple DES

Código fuente de los algoritmos utilizados por Kriptonita

TRIPLE DES

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include  <TripleDES.h>

// Constantes internas
#define KEYSETUP_ENCRYPTION     0
#define KEYSETUP_DECRYPTION     1

// Prototipos de las rutinas soportadas
void _expandPureDESKey(const WORD8*, WORD8*);
void _keysetup(WORD8*, int, WORD32*);
void _processBlock(WORD8*, WORD8*, WORD32*);

WORD32 TripleDES_GetCipherInfo
  (CIPHERINFOBLOCK* pInfo)
{
  WORD8* pSrc;
  WORD8* pDst;
  CIPHERINFOBLOCK tempinfo;
  WORD32 lI;

  // Preparación de la información de contexto
  tempinfo.lSizeOf = pInfo->lSizeOf;
  tempinfo.lBlockSize = TRIPLEDES_BLOCKSIZE;
  tempinfo.lKeySize = TRIPLEDES_KEYSIZE;
  tempinfo.blOwnHasher = BOOL_FALSE;
  tempinfo.lInitDataSize = TRIPLEDES_BLOCKSIZE;
  tempinfo.lContextSize = sizeof(TRIPLEDESCTX);
  tempinfo.bCipherIs = CIPHER_IS_BLOCKLINK;

  // Copia información
  pSrc = (WORD8*) &tempinfo;
  pDst = (WORD8*) pInfo;
  for (lI = 0; lI < tempinfo.lSizeOf; lI++)
    *pDst++ = *pSrc++;

  return CIPHER_ERROR_NOERROR;
}

WORD32 TripleDES_SelfTest
  (void* pTestContext)
{
  // Prueba el algoritmo para cifrar y descifrar
  // Valores de prueba fijos

  static WORD8 refkey[] = { 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xef };
  static WORD8 plainBlock[8] = { 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd, 0xe7 };
  static WORD8 cipherBlock[8] = { 0xc9, 0x57, 0x44, 0x25, 0x6a, 0x5e, 0xd3, 0x1d };
  WORD8 workBlock[8];
  WORD32 k[32];
```

```
  int nI;

  // Cifrado del bloque de prueba
  _keysetup(refkey, KEYSETUP_ENCRYPTION, k);
  _processBlock(plainBlock, workBlock, k);

  // Textos cifrados iguales?
  for (nI = 0; nI < 8; nI++)
    if (workBlock[nI] != cipherBlock[nI]) return CIPHER_ERROR_INVALID;
            printf("\n\n test !!!\n");
   // Descifrado del bloque de prueba
  _keysetup(refkey, KEYSETUP_DECRYPTION, k);
  _processBlock(workBlock, workBlock, k);

  // Textos claros iguales?
  for (nI = 0; nI < 8; nI++)
  {
    if (plainBlock[nI] != workBlock[nI]) return CIPHER_ERROR_INVALID;
  }
  return CIPHER_ERROR_NOERROR;
}

WORD32 TripleDES_CreateWorkContext
  (void* pContext,
   const WORD8* pKey,
   WORD32 lKeyLen,
   WORD32 lMode,
   void* pInitData,
   Cipher_RandomGenerator GetRndBytes,
   const void* pRndGenData)
{
  TRIPLEDESCTX* pCtx;
  WORD32* pCBCIV;
  int nI;

  // Creación de las tres llaves del DES partiendo de la llave original, ignorando el bit de paridad
  WORD8 deskey1[8];
  WORD8 deskey2[8];
  WORD8 deskey3[8];
  _expandPureDESKey(pKey, deskey1);
  _expandPureDESKey(pKey + 7, deskey2);
  _expandPureDESKey(pKey + 14, deskey3);

  // Inicialización de todas las llaves
  pCtx = (TRIPLEDESCTX*) pContext;

  if (lMode == CIPHER_MODE_ENCRYPT)
  {
    _keysetup(deskey1, KEYSETUP_ENCRYPTION, pCtx->k1);
    _keysetup(deskey2, KEYSETUP_DECRYPTION, pCtx->k2);
    _keysetup(deskey3, KEYSETUP_ENCRYPTION, pCtx->k3);
  }
  else
  {
    _keysetup(deskey1, KEYSETUP_DECRYPTION, pCtx->k1);
    _keysetup(deskey2, KEYSETUP_ENCRYPTION, pCtx->k2);
```

```
    _keysetup(deskey3, KEYSETUP_DECRYPTION, pCtx->k3);
  }

  // Eliminación de valores almacenados por las llaves
  for (nI = 0; nI < 8; nI++)
  {
    deskey1[nI] = 0;
    deskey2[nI] = 0;
    deskey3[nI] = 0;
  }

  // Vector IV para cifrado, en modo CBC
  pCBCIV = (WORD32*) pInitData;
  if (lMode == CIPHER_MODE_ENCRYPT)
  GetRndBytes((WORD8*) pCBCIV, 8, pRndGenData);

  //Asignación del vector
  pCtx->lCBCLo = pCBCIV[0];
  pCtx->lCBCHi = pCBCIV[1];

   return CIPHER_ERROR_NOERROR;
}

void TripleDES_ResetWorkContext
  (void* pContext,
   WORD32 lMode,
   void* pInitData,
   Cipher_RandomGenerator GetRndBytes,
   const void* pRndGenData)
{
  TRIPLEDESCTX* pCtx = (TRIPLEDESCTX*) pContext;

  // Reinicio del vector IV en CBC
  WORD32* pCBCIV = (WORD32*) pInitData;
  if (lMode == CIPHER_MODE_ENCRYPT)
  GetRndBytes((WORD8*) pCBCIV, 8, pRndGenData);
  pCtx->lCBCLo = pCBCIV[0];
  pCtx->lCBCHi = pCBCIV[1];
}

WORD32 TripleDES_DestroyWorkContext
  (void* pContext)
{
  // Inicialización de las variables de entorno
  int nI;
  WORD8* pClearIt = (WORD8*) pContext;
  for (nI = 0; nI < sizeof(TRIPLEDESCTX); nI++)
    pClearIt[nI] = 0x00;
  return CIPHER_ERROR_NOERROR;
}

void TripleDES_EncryptBuffer
  (void* pContext,
   const void* pSource,
   void* pTarget,
   WORD32 lNumOfBytes)
```

```
{
  WORD32 lNumOfInts;
  WORD32 lI;
  WORD32* pInBuf = (WORD32*) pSource;
  WORD32* pOutBuf = (WORD32*) pTarget;
  TRIPLEDESCTX* pCtx = (TRIPLEDESCTX*) pContext;

  // Calculo de las palabras de 32 bits
  lNumOfInts = lNumOfBytes >> 2;

  // Algo para cifrar?
  if (lNumOfInts < 2) return;

  // Trabajando a través de los bloques
  for (lI = 0; lI < lNumOfInts; lI+=2)
  {
    // Copia y encadena el bloque más reciente
    pOutBuf[lI]     = pInBuf[lI]     ^ pCtx->lCBCLo;
    pOutBuf[lI + 1] = pInBuf[lI + 1] ^ pCtx->lCBCHi;

    // Cifrado del bloque
    _processBlock((WORD8*) &pOutBuf[lI], (WORD8*) &pOutBuf[lI], pCtx->k1);
    _processBlock((WORD8*) &pOutBuf[lI], (WORD8*) &pOutBuf[lI], pCtx->k2);
    _processBlock((WORD8*) &pOutBuf[lI], (WORD8*) &pOutBuf[lI], pCtx->k3);

    // Asignación de valores al vector IV CBC
    pCtx->lCBCLo = pOutBuf[lI];
    pCtx->lCBCHi = pOutBuf[lI + 1];
  }
}

void TripleDES_DecryptBuffer
  (void* pContext,
   const void* pSource,
   void* pTarget,
   WORD32 lNumOfBytes,
   const void* pPreviousBlock)
{
  WORD32 lNumOfInts;
  WORD32 lI;
  WORD32 lSaveCBCLo;
  WORD32 lSaveCBCHi;
  WORD32* pInBuf = (WORD32*) pSource;
  WORD32* pOutBuf = (WORD32*) pTarget;
  WORD32* pPrevBlock = (WORD32*) pPreviousBlock;
  TRIPLEDESCTX* pCtx = (TRIPLEDESCTX*) pContext;

  // Calculo del número de palabras de 32 bloques
  lNumOfInts = lNumOfBytes >> 2;

  // Algo para descifrar?
  if (lNumOfInts < 2) return;

  // Cargo de nuevo vector CBC IV, si es neceario
  if (pPreviousBlock != CIPHER_NULL)  {
    pCtx->lCBCLo = pPrevBlock[0];
```

```
    pCtx->lCBCHi = pPrevBlock[1];
  }

  // Trabajando a través de los bloques
  for (lI = 0; lI < lNumOfInts; lI += 2)
  {

    lSaveCBCLo = pInBuf[lI];
    lSaveCBCHi = pInBuf[lI + 1];

    // Descifrado del bloque
    _processBlock((WORD8*) &pInBuf[lI], (WORD8*) &pOutBuf[lI], pCtx->k3);
    _processBlock((WORD8*) &pOutBuf[lI], (WORD8*) &pOutBuf[lI], pCtx->k2);
    _processBlock((WORD8*) &pOutBuf[lI], (WORD8*) &pOutBuf[lI], pCtx->k1);

    // Desencadenamiento del bloque más reciente
    pOutBuf[lI]     = pOutBuf[lI]     ^ pCtx->lCBCLo;
    pOutBuf[lI + 1] = pOutBuf[lI + 1] ^ pCtx->lCBCHi;

    // Nuevo vector IV
    pCtx->lCBCLo = lSaveCBCLo;
    pCtx->lCBCHi = lSaveCBCHi;
  }

}

// Rutinas de implementación
// Constantes
static WORD8 pc1[56] =
{
   56, 48, 40, 32, 24, 16,  8,  0, 57, 49, 41, 33, 25, 17,
    9,  1, 58, 50, 42, 34, 26, 18, 10,  2, 59, 51, 43, 35,
   62, 54, 46, 38, 30, 22, 14,  6, 61, 53, 45, 37, 29, 21,
   13,  5, 60, 52, 44, 36, 28, 20, 12,  4, 27, 19, 11,  3
};

static WORD8 totrot[16] =
{
   1,2,4,6,8,10,12,14,15,17,19,21,23,25,27,28
};

static WORD8 pc2[48] =
{
   13, 16, 10, 23,  0,  4,  2, 27, 14,  5, 20,  9,
   22, 18, 11,  3, 25,  7, 15,  6, 26, 19, 12,  1,
   40, 51, 30, 36, 46, 54, 29, 39, 50, 44, 32, 47,
   43, 48, 38, 55, 33, 52, 45, 41, 49, 35, 28, 31
};

static WORD16 bytebit[8] =
{
   0200, 0100, 040, 020, 010, 04, 02, 01
};

static WORD32 bigbyte[24] =
{
```

```
    0x800000L,  0x400000L,  0x200000L,  0x100000L,
    0x80000L,   0x40000L,   0x20000L,   0x10000L,
    0x8000L,    0x4000L,    0x2000L,    0x1000L,
    0x800L,     0x400L,     0x200L,     0x100L,
    0x80L,      0x40L,      0x20L,      0x10L,
    0x8L,       0x4L,       0x2L,       0x1L
};

static unsigned long SP1[64] =
{
    0x01010400L, 0x00000000L, 0x00010000L, 0x01010404L,
    0x01010004L, 0x00010404L, 0x00000004L, 0x00010000L,
    0x00000400L, 0x01010400L, 0x01010404L, 0x00000400L,
    0x01000404L, 0x01010004L, 0x01000000L, 0x00000004L,
    0x00000404L, 0x01000400L, 0x01000400L, 0x00010400L,
    0x00010400L, 0x01010000L, 0x01010000L, 0x01000404L,
    0x00010004L, 0x01000004L, 0x01000004L, 0x00010004L,
    0x00000000L, 0x00000404L, 0x00010404L, 0x01000000L,
    0x00010000L, 0x01010404L, 0x00000004L, 0x01010000L,
    0x01010400L, 0x01000000L, 0x01000000L, 0x00000400L,
    0x01010004L, 0x00010000L, 0x00010400L, 0x01000004L,
    0x00000400L, 0x00000004L, 0x01000404L, 0x00010404L,
    0x01010404L, 0x00010004L, 0x01010000L, 0x01000404L,
    0x01000004L, 0x00000404L, 0x00010404L, 0x01010400L,
    0x00000404L, 0x01000400L, 0x01000400L, 0x00000000L,
    0x00010004L, 0x00010400L, 0x00000000L, 0x01010004L
};

static unsigned long SP2[64] =
{
    0x80108020L, 0x80008000L, 0x00008000L, 0x00108020L,
    0x00100000L, 0x00000020L, 0x80100020L, 0x80008020L,
    0x80000020L, 0x80108020L, 0x80108000L, 0x80000000L,
    0x80008000L, 0x00100000L, 0x00000020L, 0x80100020L,
    0x00108000L, 0x00100020L, 0x80008020L, 0x00000000L,
    0x80000000L, 0x00008000L, 0x00108020L, 0x80100000L,
    0x00100020L, 0x80000020L, 0x00000000L, 0x00108000L,
    0x00008020L, 0x80108000L, 0x80100000L, 0x00008020L,
    0x00000000L, 0x00108020L, 0x80100020L, 0x00100000L,
    0x80008020L, 0x80100000L, 0x80108000L, 0x00008000L,
    0x80100000L, 0x80008000L, 0x00000020L, 0x80108020L,
    0x00108020L, 0x00000020L, 0x00008000L, 0x80000000L,
    0x00008020L, 0x80108000L, 0x00100000L, 0x80000020L,
    0x00100020L, 0x80008020L, 0x80000020L, 0x00100020L,
    0x00108000L, 0x00000000L, 0x80008000L, 0x00008020L,
    0x80000000L, 0x80100020L, 0x80108020L, 0x00108000L
};

static unsigned long SP3[64] =
{
    0x00000208L, 0x08020200L, 0x00000000L, 0x08020008L,
    0x08000200L, 0x00000000L, 0x00020208L, 0x08000200L,
    0x00020008L, 0x08000008L, 0x08000008L, 0x00020000L,
    0x08020208L, 0x00020008L, 0x08020000L, 0x00000208L,
    0x08000000L, 0x00000008L, 0x08020200L, 0x00000200L,
    0x00020200L, 0x08020000L, 0x08020008L, 0x00020208L,
```

```
    0x08000208L, 0x00020200L, 0x00020000L, 0x08000208L,
    0x00000008L, 0x08020208L, 0x00000200L, 0x08000000L,
    0x08020200L, 0x08000000L, 0x00020008L, 0x00000208L,
    0x00020000L, 0x08020200L, 0x08000200L, 0x00000000L,
    0x00000200L, 0x00020008L, 0x08020208L, 0x08000200L,
    0x08000008L, 0x00000200L, 0x00000000L, 0x08020008L,
    0x08000208L, 0x00020000L, 0x08000000L, 0x08020208L,
    0x00000008L, 0x00020208L, 0x00020200L, 0x08000008L,
    0x08020000L, 0x08000208L, 0x00000208L, 0x08020000L,
    0x00020208L, 0x00000008L, 0x08020008L, 0x00020200L
};

static unsigned long SP4[64] =
{
    0x00802001L, 0x00002081L, 0x00002081L, 0x00000080L,
    0x00802080L, 0x00800081L, 0x00800001L, 0x00002001L,
    0x00000000L, 0x00802000L, 0x00802000L, 0x00802081L,
    0x00000081L, 0x00000000L, 0x00800080L, 0x00800001L,
    0x00000001L, 0x00002000L, 0x00800000L, 0x00802001L,
    0x00000080L, 0x00800000L, 0x00002001L, 0x00002080L,
    0x00800081L, 0x00000001L, 0x00002080L, 0x00800080L,
    0x00002000L, 0x00802080L, 0x00802081L, 0x00000081L,
    0x00800080L, 0x00800001L, 0x00802000L, 0x00802081L,
    0x00000081L, 0x00000000L, 0x00000000L, 0x00802000L,
    0x00002080L, 0x00800080L, 0x00800081L, 0x00000001L,
    0x00802001L, 0x00002081L, 0x00002081L, 0x00000080L,
    0x00802081L, 0x00000081L, 0x00000001L, 0x00002000L,
    0x00800001L, 0x00002001L, 0x00802080L, 0x00800081L,
    0x00002001L, 0x00002080L, 0x00800000L, 0x00802001L,
    0x00000080L, 0x00800000L, 0x00002000L, 0x00802080L
};

static unsigned long SP5[64] =
{
    0x00000100L, 0x02080100L, 0x02080000L, 0x42000100L,
    0x00080000L, 0x00000100L, 0x40000000L, 0x02080000L,
    0x40080100L, 0x00080000L, 0x02000100L, 0x40080100L,
    0x42000100L, 0x42080000L, 0x00080100L, 0x40000000L,
    0x02000000L, 0x40080000L, 0x40080000L, 0x00000000L,
    0x40000100L, 0x42080100L, 0x42080100L, 0x02000100L,
    0x42080000L, 0x40000100L, 0x00000000L, 0x42000000L,
    0x02080100L, 0x02000000L, 0x42000000L, 0x00080100L,
    0x00080000L, 0x42000100L, 0x00000100L, 0x02000000L,
    0x40000000L, 0x02080000L, 0x42000100L, 0x40080100L,
    0x02000100L, 0x40000000L, 0x42080000L, 0x02080100L,
    0x40080100L, 0x00000100L, 0x02000000L, 0x42080000L,
    0x42080100L, 0x00080100L, 0x42000000L, 0x42080100L,
    0x02080000L, 0x00000000L, 0x40080000L, 0x42000000L,
    0x00080100L, 0x02000100L, 0x40000100L, 0x00080000L,
    0x00000000L, 0x40080000L, 0x02080100L, 0x40000100L
};

static unsigned long SP6[64] =
{
    0x20000010L, 0x20400000L, 0x00004000L, 0x20404010L,
    0x20400000L, 0x00000010L, 0x20404010L, 0x00400000L,
```

```
    0x20004000L, 0x00404010L, 0x00400000L, 0x20000010L,
    0x00400010L, 0x20004000L, 0x20000000L, 0x00004010L,
    0x00000000L, 0x00400010L, 0x20004010L, 0x00004000L,
    0x00404000L, 0x20004010L, 0x00000010L, 0x20400010L,
    0x20400010L, 0x00000000L, 0x00404010L, 0x20404000L,
    0x00004010L, 0x00404000L, 0x20404000L, 0x20000000L,
    0x20004000L, 0x00000010L, 0x20400010L, 0x00404000L,
    0x20404010L, 0x00400000L, 0x00004010L, 0x20000010L,
    0x00400000L, 0x20004000L, 0x20000000L, 0x00004010L,
    0x20000010L, 0x20404010L, 0x00404000L, 0x20400000L,
    0x00404010L, 0x20404000L, 0x00000000L, 0x20400010L,
    0x00000010L, 0x00004000L, 0x20400000L, 0x00404010L,
    0x00004000L, 0x00400010L, 0x20004010L, 0x00000000L,
    0x20404000L, 0x20000000L, 0x00400010L, 0x20004010L
};

static unsigned long SP7[64] =
{
    0x00200000L, 0x04200002L, 0x04000802L, 0x00000000L,
    0x00000800L, 0x04000802L, 0x00200802L, 0x04200800L,
    0x04200802L, 0x00200000L, 0x00000000L, 0x04000002L,
    0x00000002L, 0x04000000L, 0x04200002L, 0x00000802L,
    0x04000800L, 0x00200802L, 0x00200002L, 0x04000800L,
    0x04000002L, 0x04200000L, 0x04200800L, 0x00200002L,
    0x04200000L, 0x00000800L, 0x00000802L, 0x04200802L,
    0x00200800L, 0x00000002L, 0x04000000L, 0x00200800L,
    0x04000000L, 0x00200800L, 0x00200000L, 0x04000802L,
    0x04000802L, 0x04200002L, 0x04200002L, 0x00000002L,
    0x00200002L, 0x04000000L, 0x04000800L, 0x00200000L,
    0x04200800L, 0x00000802L, 0x00200802L, 0x04200800L,
    0x00000802L, 0x04000002L, 0x04200802L, 0x04200000L,
    0x00200800L, 0x00000000L, 0x00000002L, 0x04200802L,
    0x00000000L, 0x00200802L, 0x04200000L, 0x00000800L,
    0x04000002L, 0x04000800L, 0x00000800L, 0x00200002L
};

static unsigned long SP8[64] =
{
    0x10001040L, 0x00001000L, 0x00040000L, 0x10041040L,
    0x10000000L, 0x10001040L, 0x00000040L, 0x10000000L,
    0x00040040L, 0x10040000L, 0x10041040L, 0x00041000L,
    0x10041000L, 0x00041040L, 0x00001000L, 0x00000040L,
    0x10040000L, 0x10000040L, 0x10001000L, 0x00001040L,
    0x00041000L, 0x00040040L, 0x10040040L, 0x10041000L,
    0x00001040L, 0x00000000L, 0x00000000L, 0x10040040L,
    0x10000040L, 0x10001000L, 0x00041040L, 0x00040000L,
    0x00041040L, 0x00040000L, 0x10041000L, 0x00001000L,
    0x00000040L, 0x10040040L, 0x00001000L, 0x00041040L,
    0x10001000L, 0x00000040L, 0x10000040L, 0x10040000L,
    0x10040040L, 0x10000000L, 0x00040000L, 0x10001040L,
    0x00000000L, 0x10041040L, 0x00040040L, 0x10000040L,
    0x10040000L, 0x10001000L, 0x10001040L, 0x00000000L,
    0x10041040L, 0x00041000L, 0x00041000L, 0x00001040L,
    0x00001040L, 0x00040040L, 0x10000000L, 0x10041000L
};
```

```
// Asignación de una llave

void _keysetup
  (WORD8* key,
   int nKeySetup,
   WORD32* k)
{
    register int nI, nJ, nL, nM, nN;
    WORD8 pc1m[56], pcr[56];
    WORD32 kn[32];
    WORD32* raw1;
    WORD32* raw0;
    WORD32* cook;

    // configuración inicial
    for (nJ = 0; nJ < 56; nJ++)
    {
        nL = pc1[nJ];
        nM = nL & 07;
        pc1m[nJ] = (key[nL >> 3] & bytebit[nM]) ? 1 : 0;
    }
    for (nI = 0; nI < 16; nI++)
    {
        if (nKeySetup == KEYSETUP_DECRYPTION) nM = (15 - nI) << 1;
        else nM = nI << 1;
        nN = nM + 1;
        kn[nM] = kn[nN] = 0;
        for (nJ = 0; nJ < 28; nJ++)
        {
            nL = nJ + totrot[nI];
            if (nL < 28 ) pcr[nJ] = pc1m[nL];
            else pcr[nJ] = pc1m[nL - 28];
        }
        for (nJ = 28; nJ < 56; nJ++)
        {
            nL = nJ + totrot[nI];
            if (nL < 56 ) pcr[nJ] = pc1m[nL];
            else pcr[nJ] = pc1m[nL - 28];
        }
        for (nJ = 0; nJ < 24; nJ++ )
        {
            if (pcr[pc2[nJ]])      kn[nM] |= bigbyte[nJ];
            if (pcr[pc2[nJ + 24]]) kn[nN] |= bigbyte[nJ];
        }
    }

    // Almacenamiento
    raw1 = &kn[0];
    cook = k;
    for (nI = 0; nI < 16; nI++, raw1++ )
    {
      raw0 = raw1++;
      *cook    = (*raw0 & 0x00fc0000) << 6;
      *cook   |= (*raw0 & 0x00000fc0) << 10;
      *cook   |= (*raw1 & 0x00fc0000) >> 10;
      *cook++ |= (*raw1 & 0x00000fc0) >> 6;
```

```
      *cook    = (*raw0 & 0x0003f000) << 12;
      *cook   |= (*raw0 & 0x0000003f) << 16;
      *cook   |= (*raw1 & 0x0003f000) >> 4;
      *cook++ |= (*raw1 & 0x0000003f);
    }
}

void scrunch(WORD8* outof, WORD32* into)
{
    *into    = (WORD32)(*outof++) << 24;
    *into   |= (WORD32)(*outof++) << 16;
    *into   |= (WORD32)(*outof++) << 8;
    *into++ |= (WORD32)(*outof++);
    *into    = (WORD32)(*outof++) << 24;
    *into   |= (WORD32)(*outof++) << 16;
    *into   |= (WORD32)(*outof++) << 8;
    *into   |= (WORD32)(*outof);
}
void unscrunch(WORD32* outof, WORD8* into)
{
    *into++ = (WORD8)(*outof >> 24);
    *into++ = (WORD8)(*outof >> 16);
    *into++ = (WORD8)(*outof >>  8);
    *into++ = (WORD8)(*outof++);
    *into++ = (WORD8)(*outof >> 24);
    *into++ = (WORD8)(*outof >> 16);
    *into++ = (WORD8)(*outof >>  8);
    *into   = (WORD8)(*outof);
}

// Cifra o descifra el bloque de datos
// -> Bloque de entrada
// -> Bloque de salida
void _processBlock
  (WORD8* inblock,
   WORD8* outblock,
   WORD32* keys)
{
    register WORD32 lF, lWork, lRight, lLeft;
    register int nRound;
    WORD32 transblock[2];

    // Obtención del bloque
    scrunch(inblock, transblock);
    lLeft = transblock[0];
    lRight = transblock[1];

    // El bloque de entrada es dividido
    printf("\nProcesando los bloques de entrada ...\n");
    printf("============================================================\n");
    printf("\nEsta es la mitad izquierda del bloque de entrada: %X", lLeft);
    printf("\nEsta es la mitad derecha del bloque de entrada  : %X", lRight);
            printf("\n\nPresione ENTER para continuar ...\n\n");
    getchar();
    fflush(stdin);
```

```
// init. permutation
lWork = ((lLeft >> 4) ^ lRight) & 0x0f0f0f0fL;
lRight ^= lWork;
lLeft ^= (lWork << 4);

// cambio de valores en la permutación inicial
printf("\nMitad izquierda durante la permutacion inicial: %X", lLeft);
printf("\nMitad derecha durante la permutacion inicial  : %X", lRight);
printf("\n\nPresione ENTER para continuar...\n\n");
getchar();
fflush(stdin);

lWork = ((lLeft >> 16) ^ lRight) & 0x0000ffffL;
lRight ^= lWork;
lLeft ^= (lWork << 16);

// cambio de valores en la permutación inicial
printf("\nMitad izquierda durante la permutacion inicial: %X", lLeft);
printf("\nMitad derecha durante la permutacion inicial  : %X", lRight);
printf("\n\nPresione ENTER para continuar...\n\n");
getchar();
fflush(stdin);

lWork = ((lRight >> 2) ^ lLeft) & 0x33333333L;
lLeft ^= lWork;
lRight ^= (lWork << 2);

// cambio de valores en la permutación inicial
printf("\nMitad izquierda durante la permutacion inicial: %X", lLeft);
printf("\nMitad derecha durante la permutacion inicial  : %X", lRight);
printf("\n\nPresione ENTER para continuar...\n\n");
getchar();
fflush(stdin);

lWork = ((lRight >> 8) ^ lLeft) & 0x00ff00ffL;
lLeft ^= lWork;
lRight ^= (lWork << 8);

// cambio de valores en la permutación inicial
printf("\nMitad izquierda durante la permutacion inicial: %X", lLeft);
printf("\nMitad derecha durante la permutacion inicial  : %X", lRight);
printf("\n\nPresione ENTER para continuar ...\n\n");
getchar();
fflush(stdin);

lRight = ((lRight << 1) | ((lRight >> 31) & 1L)) & 0xffffffffL;
lWork = (lLeft ^ lRight) & 0xaaaaaaaaL;
lLeft ^= lWork;

// cambio de valores en la permutación inicial
printf("\nMitad izquierda durante la permutacion inicial: %X", lLeft);
printf("\nMitad derecha durante la permutacion inicial  : %X", lRight);
printf("\n\nPresione ENTER para continuar ...\n\n");
getchar();
fflush(stdin);
```

```
lRight ^= lWork;
lLeft = ((lLeft << 1) | ((lLeft >> 31) & 1L)) & 0xffffffffL;

// Bloque final después de la permutación inicial
printf("\nMitad izquierda del bloque despues de la permutacion inicial: %X", lLeft);
printf("\nMitad derecha del bloque despues de la permutacion inicial  : %X", lRight);
printf("\n\nPresione ENTER para continuar ...\n\n");
getchar();
fflush(stdin);

for (nRound = 0; nRound < 8; nRound++ )
{
   lWork  = (lRight << 28) | (lRight >> 4);
   lWork ^= *keys++;
   lF  = SP7[ lWork         & 0x3fL];
   lF |= SP5[(lWork >>  8) & 0x3fL];
   lF |= SP3[(lWork >> 16) & 0x3fL];
   lF |= SP1[(lWork >> 24) & 0x3fL];
   lWork  = lRight ^ *keys++;
   lF |= SP8[ lWork         & 0x3fL];
   lF |= SP6[(lWork >>  8) & 0x3fL];
   lF |= SP4[(lWork >> 16) & 0x3fL];
   lF |= SP2[(lWork >> 24) & 0x3fL];

   // Intercambio de los bloques de entrada
   printf("\nMitad izquierda del bloque despues de la vuelta: %X", lLeft);
   printf("\nMitad derecha del bloque despues de  la vuelta : %X", lRight);
   printf("\nLlave utilizada: %X", keys);
   printf("\n\nPresione ENTER para continuar ...\n\n");
   getchar();
   fflush(stdin);

   lLeft ^= lF;
   lWork  = (lLeft << 28) | (lLeft >> 4);
   lWork ^= *keys++;
   lF  = SP7[ lWork         & 0x3fL];
   lF |= SP5[(lWork >>  8) & 0x3fL];
   lF |= SP3[(lWork >> 16) & 0x3fL];
   lF |= SP1[(lWork >> 24) & 0x3fL];
   lWork  = lLeft ^ *keys++;
   lF |= SP8[ lWork         & 0x3fL];
   lF |= SP6[(lWork >>  8) & 0x3fL];
   lF |= SP4[(lWork >> 16) & 0x3fL];
   lF |= SP2[(lWork >> 24) & 0x3fL];
   lRight ^= lF;

   // Bloque de entrada después de una vuelta
   printf("\nMitad izquierda del bloque despues de la vuelta: %X", lLeft);
   printf("\nMitad derecha del bloque despues de la vuelta  : %X", lRight);
   printf("\nLlave utilizada: %X", keys);
        printf("\n\nPresione ENTER para continuar ...\n\n");
   getchar();
   fflush(stdin);
}

// Permutación final
```

```
    lRight = (lRight << 31) | (lRight >> 1);
    lWork = (lLeft ^ lRight) & 0xaaaaaaaaL;
    lLeft ^= lWork;
    lRight ^= lWork;
    lLeft = (lLeft << 31) | (lLeft >> 1);
    lWork = ((lLeft >> 8) ^ lRight) & 0x00ff00ffL;
    lRight ^= lWork;
    lLeft ^= (lWork << 8);
    lWork = ((lLeft >> 2) ^ lRight) & 0x33333333L;
    lRight ^= lWork;
    lLeft ^= (lWork << 2);
    lWork = ((lRight >> 16) ^ lLeft) & 0x0000ffffL;
    lLeft ^= lWork;
    lRight ^= (lWork << 16);
    lWork = ((lRight >> 4) ^ lLeft) & 0x0f0f0f0fL;
    lLeft ^= lWork;
    lRight ^= (lWork << 4);

    // Bloque de entrada después de la permutación inicial
    printf("\nMitad izquierda del bloque despues de la permutacion final: %X", lLeft);
    printf("\nMitad derecha del bloque despues de la permutacion final  : %X", lRight);
    printf("\n============================================================\n");
    printf("\nFin del bloque ...\n");
    printf("\n\nPresione ENTER para continuar ...\n\n");
    getchar();
    fflush(stdin);

    // Bloques finales
    transblock[0] = lRight;
    transblock[1] = lLeft;
    unscrunch(transblock, outblock);
}

// Expande una llave de 7 bytes a una de 8 bytes
// El bit de paridad es siempre ignorado y asignado a cero
void _expandPureDESKey
  (const WORD8* src,
   WORD8* dest)
{
  int nI, nJ, nC, nBitCounter;
  WORD8 bPureByte;
  WORD8 bBuild;

  // ciclo a través de todos los bytes
  nBitCounter = 0;
  bBuild = 0;
  nC = 0;
  for (nI = 0; nI < 7; nI++)
  {
    bPureByte = src[nI];
    for (nJ = 0; nJ < 8; nJ++)
    {
      bBuild |= (bPureByte >> nJ) & 0x01;
      bBuild <<= 1;
      nBitCounter++;
      if (nBitCounter == 7)
```

```
      {
        dest[nC++] = bBuild;
        bBuild = 0;
        nBitCounter = 0;
      }
    }
  }
}
```

RIJNDAEL

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <Rijndael.h>
#include <RijndaelBoxes.h>

typedef struct {
  WORD32 key[(RIJNDAEL_KEYSIZE / 4) * 5 + 24];
  WORD32 cbc_iv[4];
} RIJNDAELCTX;

#define rotr(x,n)   (((x) >> ((int)(n))) | ((x) << (32 - (int)(n))))
#define rotl(x,n)   (((x) << ((int)(n))) | ((x) >> (32 - (int)(n))))

#define ff_mult(a,b) (a && b ? pow_tab[(log_tab[a] + log_tab[b]) % 255] : 0)

#define byte(x,n)   ((WORD8)((x) >> (8 * (n))))

#define ls_box(x)                           \
    ((WORD32)sbx_tab[byte(x, 0)] <<  0) ^    \
    ((WORD32)sbx_tab[byte(x, 1)] <<  8) ^    \
    ((WORD32)sbx_tab[byte(x, 2)] << 16) ^    \
    ((WORD32)sbx_tab[byte(x, 3)] << 24)

#define star_x(x) (((x) & 0x7f7f7f7f) << 1) ^ ((((x) & 0x80808080) >> 7) * 0x1b)

#define imix_col(y,x)       \
    u   = star_x(x);        \
    v   = star_x(u);        \
    w   = star_x(v);        \
    t   = w ^ (x);          \
   (y)  = u ^ v ^ w;        \
   (y) ^= rotr(u ^ t,  8) ^ \
         rotr(v ^ t, 16) ^ \
         rotr(t,24)

void rijndaelSetKey(RIJNDAELCTX* pCtx,
                    const WORD8* userKey,
                    WORD32 lKeyLen,
                    WORD32 lMode)
{
  WORD32 t;
  WORD32* key = pCtx->key;
  int i;
```

```
  for (i = 0; i < RIJNDAEL_KEYSIZE / 4; i++)
    key[i] = 0;

  for (i = 0; i < RIJNDAEL_KEYSIZE; i++)
    key[i/4] |= userKey[i%lKeyLen] << ((i%4)*8);

  t = key[7];

  // Se utiliza rijndael de 256-bit
  for (i = 0; i < 7; i++)
  {
    t = rotr(t,  8);
    t = ls_box(t) ^ rco_tab[i];
    key[8 * i + 8] = t ^= key[8 * i];
    key[8 * i + 9] = t ^= key[8 * i + 1];
    key[8 * i + 10] = t ^= key[8 * i + 2];
    key[8 * i + 11] = t ^= key[8 * i + 3];
    key[8 * i + 12] = t = key[8 * i + 4] ^ ls_box(t);
    key[8 * i + 13] = t ^= key[8 * i + 5];
    key[8 * i + 14] = t ^= key[8 * i + 6];
    key[8 * i + 15] = t ^= key[8 * i + 7];
  }

  if (lMode == CIPHER_MODE_DECRYPT)
  {
    WORD32 t, u, v, w;

    for (i = 4; i < RIJNDAEL_KEYSIZE + 24; i++)
    {
      imix_col(key[i], key[i]);
    }
  }
}

#define f_rn(bo, bi, n, k)                          \
    bo[n] =  ft_tab[0][byte(bi[n],0)] ^             \
             ft_tab[1][byte(bi[(n + 1) & 3],1)] ^   \
             ft_tab[2][byte(bi[(n + 2) & 3],2)] ^   \
             ft_tab[3][byte(bi[(n + 3) & 3],3)] ^ *(k + n)

#define f_rl(bo, bi, n, k)                                  \
    bo[n] = (WORD32)sbx_tab[byte(bi[n],0)] ^                \
        rotl(((WORD32)sbx_tab[byte(bi[(n + 1) & 3],1)]),  8) ^  \
        rotl(((WORD32)sbx_tab[byte(bi[(n + 2) & 3],2)]), 16) ^  \
        rotl(((WORD32)sbx_tab[byte(bi[(n + 3) & 3],3)]), 24) ^ *(k + n)

#define f_nround(bo, bi, k) \
             f_rn(bo, bi, 0, k);             \
             f_rn(bo, bi, 1, k);             \
             f_rn(bo, bi, 2, k);             \
             f_rn(bo, bi, 3, k);             \
             k += 4

#define f_lround(bo, bi, k) \
             f_rl(bo, bi, 0, k); \
             f_rl(bo, bi, 1, k); \
```

```
        f_rl(bo, bi, 2, k); \
        f_rl(bo, bi, 3, k)

void rijndaelEncrypt(RIJNDAELCTX* pCtx,
            const WORD32* pInBlock,
            WORD32* pOutBlock)
{
  WORD32 b0[4], b1[4];
  WORD32* kp = pCtx->key;

  b0[0] = pInBlock[0] ^ *kp++;
  b0[1] = pInBlock[1] ^ *kp++;
  b0[2] = pInBlock[2] ^ *kp++;
  b0[3] = pInBlock[3] ^ *kp++;

  f_nround(b1, b0, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nLlave: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
  f_nround(b0, b1, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nLlave: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
  f_nround(b1, b0, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nLlave: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
  f_nround(b0, b1, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nLlave: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
  f_nround(b1, b0, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nLlave: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
  f_nround(b0, b1, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nLlave: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
  f_nround(b1, b0, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nLlave: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
  f_nround(b0, b1, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nLlave: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
  f_nround(b1, b0, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nLlave: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
  f_nround(b0, b1, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nLlave: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
  f_nround(b1, b0, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nLlave: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
  f_nround(b0, b1, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nLlave: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
  f_nround(b1, b0, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nLlave: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
  f_lround(b0, b1, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nLlave: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
  printf("\n\nPresione ENTER para continuar ...\n\n");
  getchar();
  fflush(stdin);

  pOutBlock[0] = b0[0];
  pOutBlock[1] = b0[1];
  pOutBlock[2] = b0[2];
  pOutBlock[3] = b0[3];
}

#define i_rn(bo, bi, n, k)                      \
   bo[n] =  it_tab[0][byte(bi[n],0)] ^           \
         it_tab[1][byte(bi[(n + 3) & 3],1)] ^   \
```

```
            it_tab[2][byte(bi[(n + 2) & 3],2)] ^   \
            it_tab[3][byte(bi[(n + 1) & 3],3)] ^ *(k + n)

#define i_rl(bo, bi, n, k)                                  \
    bo[n] = (WORD32)isb_tab[byte(bi[n],0)] ^                    \
        rotl(((WORD32)isb_tab[byte(bi[(n + 3) & 3],1)]),  8) ^  \
        rotl(((WORD32)isb_tab[byte(bi[(n + 2) & 3],2)]), 16) ^  \
        rotl(((WORD32)isb_tab[byte(bi[(n + 1) & 3],3)]), 24) ^ *(k + n)

#define i_nround(bo, bi, k) \
            i_rn(bo, bi, 0, k); \
            i_rn(bo, bi, 1, k); \
            i_rn(bo, bi, 2, k); \
            i_rn(bo, bi, 3, k); \
            k -= 4

#define i_lround(bo, bi, k) \
            i_rl(bo, bi, 0, k);  \
            i_rl(bo, bi, 1, k);  \
            i_rl(bo, bi, 2, k);  \
            i_rl(bo, bi, 3, k)

void rijndaelDecrypt(RIJNDAELCTX* pCtx,
                  const WORD32* pInBlock,
                  WORD32* pOutBlock)
{
  WORD32 b0[4], b1[4];
  WORD32* kp = pCtx->key;

  b0[0] = pInBlock[0] ^ kp[RIJNDAEL_KEYSIZE + 24];
  b0[1] = pInBlock[1] ^ kp[RIJNDAEL_KEYSIZE + 25];
  b0[2] = pInBlock[2] ^ kp[RIJNDAEL_KEYSIZE + 26];
  b0[3] = pInBlock[3] ^ kp[RIJNDAEL_KEYSIZE + 27];

  kp += RIJNDAEL_KEYSIZE + 20;

  i_nround(b1, b0, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nKey es: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
  i_nround(b0, b1, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nKey es: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
  i_nround(b1, b0, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nKey es: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
  i_nround(b0, b1, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nKey es: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
  i_nround(b1, b0, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nKey es: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
  i_nround(b0, b1, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nKey es: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
  i_nround(b1, b0, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nKey es: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
  i_nround(b0, b1, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nKey es: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
  i_nround(b1, b0, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nKey es: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
  i_nround(b0, b1, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nKey es: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
```

```
  i_nround(b1, b0, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nKey es: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
  i_nround(b0, b1, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nKey es: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
  i_nround(b1, b0, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nKey es: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
  i_lround(b0, b1, kp);printf("\nB0 es: %X %X %X %X \nB1 es: %X %X %X %X\nKey es: %X\n",
b0[0],b0[1],b0[2],b0[3], b1[0],b1[1],b1[2],b1[3], kp);
  printf("\n\nPresione ENTER para continuar ...\n\n");
  getchar();
  fflush(stdin);

  pOutBlock[0] = b0[0];
  pOutBlock[1] = b0[1];
  pOutBlock[2] = b0[2];
  pOutBlock[3] = b0[3];
}

// Funciones púbicas

WORD32 Rijndael_GetCipherInfo(CIPHERINFOBLOCK* pInfo)
{
  WORD32 lI;
  WORD8* pSrc;
  WORD8* pDst;
  CIPHERINFOBLOCK tmpInfo;

  tmpInfo.lSizeOf = pInfo->lSizeOf;
  tmpInfo.lBlockSize = RIJNDAEL_BLOCKSIZE;
  tmpInfo.lKeySize = RIJNDAEL_KEYSIZE;
  tmpInfo.blOwnHasher = BOOL_FALSE;
  tmpInfo.lInitDataSize = RIJNDAEL_BLOCKSIZE;
  tmpInfo.lContextSize = sizeof(RIJNDAELCTX);
  tmpInfo.bCipherIs = CIPHER_IS_BLOCKLINK;

  // Copia de la información del bloque
  pSrc = (WORD8*) &tmpInfo;
  pDst = (WORD8*) pInfo;
  for (lI = 0; lI < tmpInfo.lSizeOf; lI++)
    *pDst++ = *pSrc++;
  return CIPHER_ERROR_NOERROR;
}

WORD32 Rijndael_SelfTest(void* pTestContext)
{
  /*Vectores de prueba para verificar el funcionamiento del algoritmo */

  const WORD8 testkey[32] =
    { 0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6,
      0xab, 0xf7, 0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c,
      0x76, 0x2e, 0x71, 0x60, 0xf3, 0x8b, 0x4d, 0xa5,
      0x6a, 0x78, 0x4d, 0x90, 0x45, 0x19, 0x0c, 0xfe };

  const WORD8 plaintext[16] =
    { 0x32, 0x43, 0xf6, 0xa8, 0x88, 0x5a, 0x30, 0x8d,
      0x31, 0x31, 0x98, 0xa2, 0xe0, 0x37, 0x07, 0x34 };
```

```
  const WORD8 cipher_must[16] =
    { 0x1a, 0x6e, 0x6c, 0x2c, 0x66, 0x2e, 0x7d, 0xa6,
      0x50, 0x1f, 0xfb, 0x62, 0xbc, 0x9e, 0x93, 0xf3 };

  WORD32 testbuf[4];
  RIJNDAELCTX* pCtx = (RIJNDAELCTX*) pTestContext;
  int nI;

  // Inicialización del cifrado
  rijndaelSetKey(pCtx, testkey, 32, CIPHER_MODE_ENCRYPT);
  rijndaelEncrypt(pCtx, (WORD32*) plaintext, testbuf);

  // Verificación del funcionamiento de cifrado
  for (nI = 0; nI < 4; nI++)
  {
    if (testbuf[nI] != ((WORD32*) cipher_must)[nI])
      return CIPHER_ERROR_INVALID;
  }

  // Inicialización del descifrado
  rijndaelSetKey(pCtx, testkey, 32, CIPHER_MODE_DECRYPT);
  rijndaelDecrypt(pCtx, testbuf, testbuf);

  // Revisión del descifrado
  for (nI = 0; nI < 4; nI++)
  {
    if (testbuf[nI] != ((WORD32*) plaintext)[nI])
      return CIPHER_ERROR_INVALID;
  }

  return CIPHER_ERROR_NOERROR;
}

WORD32 Rijndael_CreateWorkContext(void* pContext,
                          const WORD8* pKey,
                          WORD32 lKeyLen,
                          WORD32 lMode,
                          void* pInitData,
                          Cipher_RandomGenerator GetRndBytes,
                          const void* pRandGenData)
{
  RIJNDAELCTX* pCtx = (RIJNDAELCTX*) pContext;
  WORD32* pCBCIV;
  // Inicialización de llaves
  rijndaelSetKey(pCtx, pKey, lKeyLen, lMode);

  pCBCIV = (WORD32*) pInitData;
  if (lMode == CIPHER_MODE_ENCRYPT)
    GetRndBytes((WORD8*) pCBCIV, RIJNDAEL_BLOCKSIZE, pRandGenData);

  // Asignar IV en modo CBC
  pCtx->cbc_iv[0] = pCBCIV[0];
  pCtx->cbc_iv[1] = pCBCIV[1];
  pCtx->cbc_iv[2] = pCBCIV[2];
  pCtx->cbc_iv[3] = pCBCIV[3];
```

```
 return CIPHER_ERROR_NOERROR;
}

void Rijndael_ResetWorkContext(void* pContext,
                        WORD32 lMode,
                        void* pInitData,
                        Cipher_RandomGenerator GetRndBytes,
                        const void* pRandGenData)
{
 RIJNDAELCTX* pCtx = (RIJNDAELCTX*) pContext;
 WORD32* pCBCIV = (WORD32*) pInitData;

 if (lMode == CIPHER_MODE_ENCRYPT)
   GetRndBytes((WORD8*) pCBCIV, RIJNDAEL_BLOCKSIZE, pRandGenData);

 pCtx->cbc_iv[0] = pCBCIV[0];
 pCtx->cbc_iv[1] = pCBCIV[1];
 pCtx->cbc_iv[2] = pCBCIV[2];
 pCtx->cbc_iv[3] = pCBCIV[3];
}

WORD32 Rijndael_DestroyWorkContext(void* pContext)
{
 int nI;
 WORD8* pCtxBuf = (WORD8*) pContext;

 for (nI = 0; nI < sizeof(RIJNDAELCTX); nI++)
   pCtxBuf[nI] = 0x00;

 return CIPHER_ERROR_NOERROR;
}

void Rijndael_EncryptBuffer(void* pContext,
                       const void*  pSource,
                       void* pTarget,
                       WORD32 lNumOfBytes)
{
 WORD32 lNumOfBlocks;
 WORD32* pInBuf = (WORD32*) pSource;
 WORD32* pOutBuf = (WORD32*) pTarget;
 RIJNDAELCTX* pCtx = (RIJNDAELCTX*) pContext;

 lNumOfBlocks = lNumOfBytes / RIJNDAEL_BLOCKSIZE;

 while (lNumOfBlocks--)
 {
   pOutBuf[0] = pInBuf[0] ^ pCtx->cbc_iv[0];
   pOutBuf[1] = pInBuf[1] ^ pCtx->cbc_iv[1];
   pOutBuf[2] = pInBuf[2] ^ pCtx->cbc_iv[2];
   pOutBuf[3] = pInBuf[3] ^ pCtx->cbc_iv[3];

   // Cifrado del contenido del buffer
   rijndaelEncrypt(pCtx, pOutBuf, pOutBuf);

   // Asignar un nuevo IV
```

```
    pCtx->cbc_iv[0] = pOutBuf[0];
    pCtx->cbc_iv[1] = pOutBuf[1];
    pCtx->cbc_iv[2] = pOutBuf[2];
    pCtx->cbc_iv[3] = pOutBuf[3];

    pInBuf  += RIJNDAEL_BLOCKSIZE / 4;
    pOutBuf += RIJNDAEL_BLOCKSIZE / 4;
   }
  }

void Rijndael_DecryptBuffer(void* pContext,
                        const void* pSource,
                        void*  pTarget,
                        WORD32 lNumOfBytes,
                        const void* pPreviousBlock)
{
  WORD32 lNumOfBlocks;
  WORD32* pInBuf = (WORD32*) pSource;
  WORD32* pOutBuf = (WORD32*) pTarget;
  WORD32* pPrevBlock = (WORD32*) pPreviousBlock;
  WORD32 save_cbc_iv[4];
  RIJNDAELCTX* pCtx = (RIJNDAELCTX*) pContext;

  lNumOfBlocks = lNumOfBytes / RIJNDAEL_BLOCKSIZE;

  // Cargar nuevos vectores IV, si es necesario
  if (pPreviousBlock != CIPHER_NULL)
  {
    pCtx->cbc_iv[0] = pPrevBlock[0];
    pCtx->cbc_iv[1] = pPrevBlock[1];
    pCtx->cbc_iv[2] = pPrevBlock[2];
    pCtx->cbc_iv[3] = pPrevBlock[3];
  }

  while (lNumOfBlocks--)
  {
    // Almacenamiento del vector IV
    save_cbc_iv[0] = pInBuf[0];
    save_cbc_iv[1] = pInBuf[1];
    save_cbc_iv[2] = pInBuf[2];
    save_cbc_iv[3] = pInBuf[3];

    // Descifrado del bloque
    rijndaelDecrypt(pCtx, pInBuf, pOutBuf);

    // Liberación del bloque
    pOutBuf[0] ^= pCtx->cbc_iv[0];
    pOutBuf[1] ^= pCtx->cbc_iv[1];
    pOutBuf[2] ^= pCtx->cbc_iv[2];
    pOutBuf[3] ^= pCtx->cbc_iv[3];

    // Asignación del nuevo vector IV
    pCtx->cbc_iv[0] = save_cbc_iv[0];
    pCtx->cbc_iv[1] = save_cbc_iv[1];
    pCtx->cbc_iv[2] = save_cbc_iv[2];
    pCtx->cbc_iv[3] = save_cbc_iv[3];
```

```
    pInBuf  += RIJNDAEL_BLOCKSIZE / 4;
    pOutBuf += RIJNDAEL_BLOCKSIZE / 4;
  }
}
```

IDEA

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <IDEA.h>

typedef struct
{
  WORD16 key[IDEA_KEYLEN];
  WORD32 lCBCHi;
  WORD32 lCBCLo;
}
IDEACTX;

void ideaCipher(WORD8[8], WORD8[8], WORD16*);
void ideaExpandKey(WORD8*, WORD16*);
void ideaInvertKey(WORD16*, WORD16[]);

WORD32 IDEA_GetCipherInfo
  (CIPHERINFOBLOCK* pInfo)
{
  WORD32 lI;
  WORD8* pSrc;
  WORD8* pDst;
  CIPHERINFOBLOCK tempinfo;

  // Preparación de la información de contexto
  tempinfo.lSizeOf = pInfo->lSizeOf;
  tempinfo.lBlockSize = IDEA_BLOCKSIZE;
  tempinfo.lKeySize = IDEA_KEYSIZE;
  tempinfo.blOwnHasher = BOOL_FALSE;
  tempinfo.lInitDataSize = IDEA_BLOCKSIZE;
  tempinfo.lContextSize = sizeof(IDEACTX);
  tempinfo.bCipherIs = CIPHER_IS_BLOCKLINK;

  // Copia de los bytes
  pSrc = (WORD8*) &tempinfo;
  pDst = (WORD8*) pInfo;
  for (lI = 0; lI < tempinfo.lSizeOf; lI++)
    *pDst++ = *pSrc++;

  return CIPHER_ERROR_NOERROR;
}

WORD32 IDEA_SelfTest
  (void* pTestContext)
{
  // Prueba de funcionamiento del algoritmo, cifrado y descifrado
```

```
  static WORD8 testkey[IDEA_KEYSIZE] = { 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16 } ;
  WORD16 key[IDEA_KEYLEN];

  static WORD32 plainBlock[2] = { 0x1234567, 0x89abcdef };
  WORD32 cipherBlock[2];

  // Asignando llaves débiles
  ideaExpandKey(testkey, key);

  // Cifrado del bloque de prueba
  ideaCipher((WORD8*) plainBlock, (WORD8*) cipherBlock, key);

  // Creación de una llave de cifrado
  ideaInvertKey(key, key);

  // Descifrado del bloque de prueba
  ideaCipher((WORD8*) cipherBlock, (WORD8*) cipherBlock, key);

  // Textos claros iguales?
  return ((plainBlock[0] == cipherBlock[0]) &&
         (plainBlock[1] == cipherBlock[1])) ?
        CIPHER_ERROR_NOERROR : CIPHER_ERROR_INVALID;
}

WORD32 IDEA_CreateWorkContext
  (void* pContext,
   const WORD8* pKey,
   WORD32 lKeyLen,
   WORD32 lMode,
   void* pInitData,
   Cipher_RandomGenerator GetRndBytes,
   const void* pRndGenData)
{
  IDEACTX* pCtx = (IDEACTX*) pContext;
  WORD32* pCBCIV = (WORD32*) pInitData;

  // Expansión de la llave
  ideaExpandKey((WORD8*) pKey,
              (WORD16*) &(pCtx->key));

  // Inversa de la llave
  if (lMode == CIPHER_MODE_DECRYPT)
  {
    ideaInvertKey((WORD16*) &(pCtx->key),
                (WORD16*) &(pCtx->key));
  }

  // Para el cifrado se crea el vector IV en modo CBC
  if (lMode == CIPHER_MODE_ENCRYPT)
  {
    GetRndBytes((WORD8*) pCBCIV, 8, pRndGenData);
  }

  // Asignar el vector, nuevo o no
  pCtx->lCBCLo = pCBCIV[0];
  pCtx->lCBCHi = pCBCIV[1];
```

```
  // FIXME: weak keys in IDEA?
  return CIPHER_ERROR_NOERROR;
}

void IDEA_ResetWorkContext
  (void* pContext,
   WORD32 lMode,
   void* pInitData,
   Cipher_RandomGenerator GetRndBytes,
   const void* pRndGenData)
{
  IDEACTX* pCtx = (IDEACTX*) pContext;

  // Reinicio del vector IV en modo CBC

  WORD32* pCBCIV = (WORD32*) pInitData;

  if (lMode == CIPHER_MODE_ENCRYPT)
  {
    GetRndBytes((WORD8*) pCBCIV, 8, pRndGenData);
  }

  pCtx->lCBCLo = pCBCIV[0];
  pCtx->lCBCHi = pCBCIV[1];

}

WORD32 IDEA_DestroyWorkContext
  (void* pContext)
{
  // Limpieza de contexto
  int nI;
  WORD8* clearIt = (WORD8*) pContext;

  for (nI = 0; nI < sizeof(IDEACTX); nI++)
  {
    clearIt[nI] = 0x00;
  }

  return CIPHER_ERROR_NOERROR;
}

void IDEA_EncryptBuffer
  (void* pContext,
   const void* pSource,
   void* pTarget,
   WORD32 lNumOfBytes)
{
  WORD32 lNumOfInts;
  WORD32 lI;
  WORD32* pInBuf = (WORD32*) pSource;
  WORD32* pOutBuf = (WORD32*) pTarget;
  IDEACTX* pCtx = (IDEACTX*) pContext;

  // Cálculo del número de palabras de 32 bits
```

```
  lNumOfInts = lNumOfBytes >> 2;

  // Algo para cifrar?
  if (lNumOfInts < 2) return;

  // Trabajando a través de los bloques...
  for (lI = 0; lI < lNumOfInts; lI += 2)
  {
    // Copia y encadena el bloque actual
    pOutBuf[lI]     = pInBuf[lI]     ^ pCtx->lCBCLo;
    pOutBuf[lI + 1] = pInBuf[lI + 1] ^ pCtx->lCBCHi;

    // Cifra el bloque
    ideaCipher((WORD8*) &(pOutBuf[lI]),
               (WORD8*) &(pOutBuf[lI]),
               pCtx->key);

    // Asigna el nuevo vector IV en modo CBC
    pCtx->lCBCLo = pOutBuf[lI];
    pCtx->lCBCHi = pOutBuf[lI + 1];
  }
}

void IDEA_DecryptBuffer
  (void* pContext,
   const void* pSource,
   void* pTarget,
   WORD32 lNumOfBytes,
   const void* pPreviousBlock)
{
  WORD32 lNumOfInts;
  WORD32 lI;
  WORD32 lSaveCBCLo;
  WORD32 lSaveCBCHi;
  WORD32* pInBuf = (WORD32*) pSource;
  WORD32* pOutBuf = (WORD32*) pTarget;
  WORD32* pPrevBlock = (WORD32*) pPreviousBlock;
  IDEACTX* pCtx = (IDEACTX*) pContext;

  // Cálculo del número de palabras de 32 bits
  lNumOfInts = lNumOfBytes >> 2;

  // Algo para descifrar?
  if (lNumOfInts < 2) return;

  // Cargar el nuevo vector IV CBC, si es neceario
  if (pPreviousBlock != CIPHER_NULL)
  {
    pCtx->lCBCLo = pPrevBlock[0];
    pCtx->lCBCHi = pPrevBlock[1];
  }

  // Trabajando a traves de todos los bloques...
  for (lI = 0; lI < lNumOfInts; lI += 2)
  {
    // Almacenamiento de IV en modo CBC
```

```
    lSaveCBCLo = pInBuf[lI];
    lSaveCBCHi = pInBuf[lI + 1];

    // Descifrado del bloque
    ideaCipher((WORD8*) &(pInBuf[lI]),
            (WORD8*) &(pOutBuf[lI]),
            pCtx->key);

    // Desencadenamiento del bloque
    pOutBuf[lI]     = pOutBuf[lI]     ^ pCtx->lCBCLo;
    pOutBuf[lI + 1] = pOutBuf[lI + 1] ^ pCtx->lCBCHi;

    // Asignación del bloque en el vector IV modo CBC
    pCtx->lCBCLo = lSaveCBCLo;
    pCtx->lCBCHi = lSaveCBCHi;
  }
}

#define low16(x) ((x) & 0x0ffff)
#define MUL(x,y) ( x = low16(x-1), t16 = low16((y)-1), \
                t32 = (WORD32)x*t16 + x + t16 + 1, x = (WORD16)(low16(t32)), \
                t16 = (WORD16)(t32>>16), x = (x-t16) + (x<t16) )

void ideaCipher
  (WORD8 inbuf[8],
   WORD8 outbuf[8],
   WORD16 *key)
{
   register WORD16 wX1, wX2, wX3, wX4, wS2, wS3;
   register WORD16 t16;
   register WORD32 t32;
   WORD16 *in, *out;

   // get the block
   in = (WORD16*) inbuf;
   wX1 = in[0];  wX2 = in[1];
   wX3 = in[2];  wX4 = in[3];

   // Vuelta #1
   printf("\nLlaves utilizadas: \n%X\n%X\n%X\n%X\n%X\n%X\n",
key[0],key[1],key[2],key[3],key[4],key[5]);
   MUL(wX1, key[0]);
   wX2 += key[1];
   wX3 += key[2];
   MUL(wX4, key[3]);
   wS3 = wX3;
   wX3 ^= wX1;
   MUL(wX3, key[4]);
   wS2 = wX2;
   wX2 ^= wX4;
   wX2 += wX3;
   MUL(wX2, key[5]);
   wX3 += wX2;
   wX1 ^= wX2;  wX4 ^= wX3;
   wX2 ^= wS3;  wX3 ^= wS2;
```

```
    // Bloque de entrada después de la primera vuelta
    printf("\nX1 Despues de la primera vuelta: %X", wX1);
    printf("\nX2 Despues de la primera vuelta: %X", wX2);
    printf("\nX3 Despues de la primera vuelta: %X", wX3);
    printf("\nX4 Despues de la primera vuelta: %X", wX4);
    printf("\n\n Presione ENTER para continuar ...\n\n");
    getchar();
    fflush(stdin);

    // Vuelta #2
    printf("\nLlaves utilizadas: \n%X\n%X\n%X\n%X\n%X\n%X\n",
key[6],key[7],key[8],key[9],key[10],key[11]);
    MUL(wX1, key[6]);
    wX2 += key[7];
    wX3 += key[8];
    MUL(wX4, key[9]);
    wS3 = wX3;
    wX3 ^= wX1;
    MUL(wX3, key[10]);
    wS2 = wX2;
    wX2 ^= wX4;
    wX2 += wX3;
    MUL(wX2, key[11]);
    wX3 += wX2;
    wX1 ^= wX2;  wX4 ^= wX3;
    wX2 ^= wS3;  wX3 ^= wS2;

    // Bloque de entrada después de una vuelta
    printf("\nX1 Despues de la segunda vuelta: %X", wX1);
    printf("\nX2 Despues de la segunda vuelta: %X", wX2);
    printf("\nX3 Despues de la segunda vuelta: %X", wX3);
    printf("\nX4 Despues de la segunda vuelta: %X", wX4);
    printf("\n\n Presione ENTER para continuar ...\n\n");
    getchar();
    fflush(stdin);

    // Vuelta#3
    printf("\nLlaves utilizadas: \n%X\n%X\n%X\n%X\n%X\n%X\n",
key[12],key[13],key[14],key[15],key[16],key[17]);
    MUL(wX1, key[12]);
    wX2 += key[13];
    wX3 += key[14];
    MUL(wX4, key[15]);
    wS3 = wX3;
    wX3 ^= wX1;
    MUL(wX3, key[16]);
    wS2 = wX2;
    wX2 ^= wX4;
    wX2 += wX3;
    MUL(wX2, key[17]);
    wX3 += wX2;
    wX1 ^= wX2;  wX4 ^= wX3;
    wX2 ^= wS3;  wX3 ^= wS2;
    // Bloque de entrada después de una vuelta
    printf("\nX1 Despues de la tercera vuelta: %X", wX1);
    printf("\nX2 Despues de la tercera vuelta: %X", wX2);
```

```
    printf("\nX3 Despues de la tercera vuelta: %X", wX3);
    printf("\nX4 Despues de la tercera vuelta: %X", wX4);
    printf("\n\n Presione ENTER para continuar ...\n\n");
    getchar();
    fflush(stdin);

    // Vuelta #4
    printf("\nLlaves utilizadas: \n%X\n%X\n%X\n%X\n%X\n%X\n%X\n",
key[18],key[19],key[20],key[21],key[22],key[23]);
    MUL(wX1, key[18]);
    wX2 += key[19];
    wX3 += key[20];
    MUL(wX4, key[21]);
    wS3 = wX3;
    wX3 ^= wX1;
    MUL(wX3, key[22]);
    wS2 = wX2;
    wX2 ^= wX4;
    wX2 += wX3;
    MUL(wX2, key[23]);
    wX3 += wX2;
    wX1 ^= wX2;  wX4 ^= wX3;
    wX2 ^= wS3;  wX3 ^= wS2;
    // Bloque de entrada después de una vuelta
    printf("\nX1 Despues de la cuarta vuelta: %X", wX1);
    printf("\nX2 Despues de la cuarta vuelta: %X", wX2);
    printf("\nX3 Despues de la cuarta vuelta: %X", wX3);
    printf("\nX4 Despues de la cuarta vuelta: %X", wX4);
    printf("\n\n Presione ENTER para continuar ...\n\n");
    getchar();
    fflush(stdin);

    // Vuelta #5
    printf("\nLlaves utilizadas: \n%X\n%X\n%X\n%X\n%X\n%X\n%X\n",
key[24],key[25],key[26],key[27],key[28],key[29]);
    MUL(wX1, key[24]);
    wX2 += key[25];
    wX3 += key[26];
    MUL(wX4, key[27]);
    wS3 = wX3;
    wX3 ^= wX1;
    MUL(wX3, key[28]);
    wS2 = wX2;
    wX2 ^= wX4;
    wX2 += wX3;
    MUL(wX2, key[29]);
    wX3 += wX2;
    wX1 ^= wX2;  wX4 ^= wX3;
    wX2 ^= wS3;  wX3 ^= wS2;
    // Bloque de entrada después de una vuelta
    printf("\nX1 Despues de la quinta vuelta: %X", wX1);
    printf("\nX2 Despues de la quinta vuelta: %X", wX2);
    printf("\nX3 Despues de la quinta vuelta: %X", wX3);
    printf("\nX4 Despues de la quinta vuelta: %X", wX4);
    printf("\n\n Presione ENTER para continuar...\n\n");
    getchar();
```

```
    fflush(stdin);

    // Vuelta #6
    printf("\n Llaves utilizadas: \n%X\n%X\n%X\n%X\n%X\n%X\n",
key[30],key[31],key[32],key[33],key[34],key[35]);
    MUL(wX1, key[30]);
    wX2 += key[31];
    wX3 += key[32];
    MUL(wX4, key[33]);
    wS3 = wX3;
    wX3 ^= wX1;
    MUL(wX3, key[34]);
    wS2 = wX2;
    wX2 ^= wX4;
    wX2 += wX3;
    MUL(wX2, key[35]);
    wX3 += wX2;
    wX1 ^= wX2;  wX4 ^= wX3;
    wX2 ^= wS3;  wX3 ^= wS2;
    // Bloque de entrada después de una vuelta
    printf("\nX1 Despues de la sexta vuelta: %X", wX1);
    printf("\nX2 Despues de la sexta vuelta: %X", wX2);
    printf("\nX3 Despues de la sexta vuelta: %X", wX3);
    printf("\nX4 Despues de la sexta vuelta: %X", wX4);
    printf("\n\n Presione ENTER para continuar ...\n\n");
    getchar();
    fflush(stdin);

    // Vuelta #7
    printf("\nLlaves utilizadas: \n%X\n%X\n%X\n%X\n%X\n%X\n%X\n",
key[36],key[37],key[38],key[39],key[40],key[41]);
    MUL(wX1, key[36]);
    wX2 += key[37];
    wX3 += key[38];
    MUL(wX4, key[39]);
    wS3 = wX3;
    wX3 ^= wX1;
    MUL(wX3, key[40]);
    wS2 = wX2;
    wX2 ^= wX4;
    wX2 += wX3;
    MUL(wX2, key[41]);
    wX3 += wX2;
    wX1 ^= wX2;  wX4 ^= wX3;
    wX2 ^= wS3;  wX3 ^= wS2;
    // Bloque de entrada después de una vuelta
    printf("\nX1 Despues de la septima vuelta: %X", wX1);
    printf("\nX2 Despues de la septima vuelta: %X", wX2);
    printf("\nX3 Despues de la septima vuelta: %X", wX3);
    printf("\nX4 Despues de la septima vuelta: %X", wX4);
    printf("\n\n Presione ENTER para continuar ...\n\n");
    getchar();
    fflush(stdin);

    // Vuelta #8
```

```
   printf("\nLlaves utilizadas: \n%X\n%X\n%X\n%X\n%X\n%X\n",
key[42],key[43],key[44],key[45],key[46],key[47]);
   MUL(wX1, key[42]);
   wX2 += key[43];
   wX3 += key[44];
   MUL(wX4, key[45]);
   wS3 = wX3;
   wX3 ^= wX1;
   MUL(wX3, key[46]);
   wS2 = wX2;
   wX2 ^= wX4;
   wX2 += wX3;
   MUL(wX2, key[47]);
   wX3 += wX2;
   wX1 ^= wX2;  wX4 ^= wX3;
   wX2 ^= wS3;  wX3 ^= wS2;

   // Bloque de entrada después de una vuelta
   printf("\nX1 Despues de la octava avuelta: %X", wX1);
   printf("\nX2 Despues de la octava avuelta: %X", wX2);
   printf("\nX3 Despues de la octava avuelta: %X", wX3);
   printf("\nX4 Despues de la octava avuelta: %X", wX4);
   printf("\n\n Presione ENTER para continuar ...\n\n");
   getchar();
   fflush(stdin);

   // Semi vuelta final
   printf("\nLlaves utilizadas: \n%X\n%X\n%X\n%X\n", key[48],key[49],key[50],key[51]);
   MUL(wX1, key[48]);
   wX3 += key[49];
   wX2 += key[50];
   MUL(wX4, key[51]);

   // Bloque de entrada después de una vuelta
   printf("\nX1 Despues de la semi vuelta final: %X", wX1);
   printf("\nX2 Despues de la semi vuelta final: %X", wX2);
   printf("\nX3 Despues de la semi vuelta final: %X", wX3);
   printf("\nX4 Despues de la semi vuelta final: %X", wX4);
   printf("\n\n Presione ENTER para continuar ...\n\n");
   getchar();
   fflush(stdin);

   // Presentación del bloque cifrado
   out = (WORD16*) outbuf;
   *out++ = wX1;
   *out++ = wX3;
   *out++ = wX2;
   *out = wX4;
}

WORD16 mulInv
  (WORD16 wX)
{
   WORD16 wT0, wT1, wQ, wY;

   // o y 1 como auto inversas
```

```
    if (wX <= 1) return wX;

    // ya que wX >= 2, esto se cabe en 16 bits
    wT1 = 0x010001L / wX;
    wY = 0x010001L % wX;

    if (wY == 1)
    {
      return (WORD16) low16(1 - wT1);
    }
    wT0 = 1;

    do {
      wQ = wX / wY;
      wX = wX % wY;
      wT0 += wQ * wT1;
      if (wX == 1)  return wT0;
      wQ = wY / wX;
      wY = wY % wX;
      wT1 += wQ * wT0;
    }
    while (wY != 1);

    return (WORD16) low16(1 - wT1);
}

void ideaExpandKey
  (WORD8* userkey,
   WORD16* ek)
{
    int nI, nJ;

    // Expansión de la llave de 128 bits para una llave de cifrado
    for (nJ = 0; nJ < 8; nJ++)
    {
      ek[nJ] = (userkey[0] << 8) + userkey[1];
      userkey += 2;
    }

    for (nI=0; nJ < IDEA_KEYLEN; nJ++)
    {
      nI++;
      ek[nI + 7] = (ek[nI & 7] << 9) | (ek[nI + 1 & 7] >> 7);
      ek += nI & 8;
      nI &= 7;
    }
}

#define NEG(x) (- (int) (x))

void ideaInvertKey
  (WORD16* ek,
   WORD16 dk[IDEA_KEYLEN])
{
    WORD16 temp[IDEA_KEYLEN];
    register int nK, nP, nR;
```

```
  // Inversión de la llave para descifrado
  nP = IDEA_KEYLEN;
  temp[nP-1] = mulInv(ek[3]);
  temp[nP-2] = NEG(ek[2]);
  temp[nP-3] = NEG(ek[1]);
  temp[nP-4] = mulInv(ek[0]);

  nK = 4;
  nP -= 4;

  for (nR = IDEA_ROUNDS - 1; nR > 0; nR--)
  {
   temp [nP-1] = ek[nK+1];
   temp [nP-2] = ek[nK];
   temp [nP-3] = mulInv(ek[nK+5]);
   temp [nP-4] = NEG(ek[nK+3]);
   temp [nP-5] = NEG(ek[nK+4]);
   temp [nP-6] = mulInv(ek[nK+2]);
   nK += 6; nP -= 6;
  }

  temp [nP-1] = ek[nK+1];
  temp [nP-2] = ek[nK];
  temp [nP-3] = mulInv(ek[nK+5]);
  temp [nP-4] = NEG(ek [nK+4]);
  temp [nP-5] = NEG(ek [nK+3]);
  temp [nP-6] = mulInv(ek[nK+2]);

  for (nK = 0; nK < IDEA_KEYLEN; nK++)
  {
   dk[nK] = temp[nK];
   temp[nK] = 0;
  }
}
```

Librerias utilizadas

TRIPLE DES.H

```
#ifndef __TRIPLEDES_H
#define __TRIPLEDES_H
#include "CipherDef.h"

#define TRIPLEDES_KEYSIZE      21   // 3 * (8 - 1)
#define TRIPLEDES_BLOCKSIZE     8
#define TRIPLEDES_CIPHERNAME    "triple-DES"

typedef struct {

  WORD32 k1[32];
  WORD32 k2[32];
  WORD32 k3[32];

  WORD32 lCBCLo;
  WORD32 lCBCHi;
}
TRIPLEDESCTX;

WORD32 TripleDES_GetCipherInfo(CIPHERINFOBLOCK*);

WORD32 TripleDES_SelfTest (void*);

WORD32 TripleDES_CreateWorkContext(void*, const WORD8*, WORD32, WORD32, void*,
                            Cipher_RandomGenerator, const void*);

void TripleDES_ResetWorkContext(void*, WORD32, void*,
                          Cipher_RandomGenerator, const void*);

WORD32 TripleDES_DestroyWorkContext (void*);

void TripleDES_EncryptBuffer(void*, const void*, void*, WORD32);

void TripleDES_DecryptBuffer(void*, const void*, void*, WORD32, const void*);

#ifdef __cplusplus
}
#endif
#endif
```

Rijndael.h

```
#ifndef __RIJNDAEL_H
#define __RIJNDAEL_H
#ifdef __cplusplus
extern "C" {
#endif
#include "CipherDef.h"

#define RIJNDAEL_KEYSIZE    32
#define RIJNDAEL_BLOCKSIZE  16
#define RIJNDAEL_CIPHERNAME "Rijndael"
```

```
WORD32 Rijndael_GetCipherInfo(CIPHERINFOBLOCK*);
WORD32 Rijndael_SelfTest(void*);
WORD32 Rijndael_CreateWorkContext(void*,
                          const WORD8*,
                          WORD32,
                          WORD32,
                          void*,
                          Cipher_RandomGenerator,
                          const void*);

void Rijndael_ResetWorkContext(void*,
                        WORD32,
                        void*,
                        Cipher_RandomGenerator,
                        const void*);

WORD32 Rijndael_DestroyWorkContext(void*);

void Rijndael_EncryptBuffer(void*,
                      const void*,
                      void*,
                      WORD32);

void Rijndael_DecryptBuffer(void*,
                      const void*,
                      void*,
                      WORD32,
                      const void*);

#ifdef __cplusplus
}
#endif
#endif
```

IDEA.H

```
#ifndef __IDEA_H
#define __IDEA_H

#ifdef __cplusplus
extern "C" {
#endif
#include "CipherDef.h"

#define IDEA_KEYSIZE        16
#define IDEA_BLOCKSIZE      8
#define IDEA_ROUNDS         8
#define IDEA_KEYLEN         (6 * IDEA_ROUNDS + 4)
#define IDEA_CIPHERNAME     "IDEA"

WORD32 IDEA_GetCipherInfo(CIPHERINFOBLOCK*);

WORD32 IDEA_SelfTest (void*);

WORD32 IDEA_CreateWorkContext(void*, const WORD8*, WORD32, WORD32, void*,
```

```
                    Cipher_RandomGenerator, const void*);

void IDEA_ResetWorkContext(void*, WORD32, void*,
                    Cipher_RandomGenerator, const void*);

WORD32 IDEA_DestroyWorkContext (void*);

void IDEA_EncryptBuffer(void*, const void*, void*, WORD32);

void IDEA_DecryptBuffer(void*, const void*, void*, WORD32, const void*);

#ifdef __cplusplus
}
#endif
#endif
```