IMPLEMENTACION DE HERRAMIENTAS CRIPTOGRAFICAS DE CLAVE PÚBLICA Y FUNCIONES HASH PARA LA CUTB

CARLOS SIERRA CASTILLO
JOSÉ CARLOS MONTES NARANJO

CORPORACIÓN UNIVERSITARIA TECNOLÓGICA DE BOLÍVAR
FACULTAD DE INGENIERÍA DE SISTEMAS
CARTAGENA, D.T.
1999

IMPLEMENTACION DE HERRAMIENTAS CRIPTOGRAFICAS DE CLAVE PÚBLICA Y FUNCIONES HASH PARA LA CUTB

CARLOS SIERRA CASTILLO
JOSÉ CARLOS MONTES NARANJO

Trabajo de Grado para optar el título de
Ingenieros de Sistemas

DIRECTOR
JAIME ARCILA IRIARTE
Ingeniero Eléctrico

ASESORES
GIOVANI VASQUEZ
EDUARDO CONCEPCIÓN

CORPORACIÓN UNIVERSITARIA TECNOLÓGICA DE BOLÍVAR
FACULTAD DE INGENIERÍA DE SISTEMAS
CARTAGENA, D.T.
1999

Cartagena, 9 de Diciembre de 1999

Ingeniero
JUAN CARLOS MANTILLA
Decano Facultad de Ingeniería de Sistemas
Corporación Universitaria Tecnológica de Bolívar
Ciudad.

Respetado Ingeniero:

Por medio de la presente me permito informarles que he dirigido a las estudiantes **CARLOS SIERRA** Y **JOSE CARLOS MONTES NARANJO,** en el proyecto de grado titulado**: "IMPLEMENTACION DE HERRAMIENTAS CRIPTOGRAFICAS DE CLAVE PUBLICA Y FUNCIONES HASH PARA LA CUTB**", presentado como requisito para optar el título de Ingeniero de Sistemas.

Agradeciendo la atención prestada.

JAIME ARCILA IRIARTE
Ingeniero Eléctrico

Cartagena, 9 de Diciembre de 1999

Ingeniero
JUAN CARLOS MANTILLA
Decano Facultad de Ingeniería de Sistemas
Corporación Universitaria Tecnológica de Bolívar
Ciudad.

Respetado Ingeniero:

Por medio de la presente ponemos a su consideración el Trabajo de Grado titulado **"IMPLEMENTACION DE HERRAMIENTAS CRIPTOGRAFICAS DE CLAVE PUBLICA Y FUNCIONES HASH PARA LA CUTB**", **p**ara su correspondiente evaluación

El desarrollo de dicho proyecto se llevo a cabo como requisito para obtener el título de **INGENIERO DE SISTEMAS**.

Agradeciendo de antemano su gentil colaboración y apoyo.

Cordialmente,

CARLOS SIERRA CASTILLO
Cod: 9305 133

JOSE CARLOS MONTES NARANJO
Cod:9305 550

Cartagena, 9 de Diciembre de 1999

Ingeniera

VILMA VIVIANA OJEDA

Asistente de Investigaciones

Corporación Universitaria Tecnológica de Bolívar

Ciudad.

Respetada Ingeniera:

Por medio de la presente ponemos a su consideración el Trabajo de Grado titulado **"IMPLEMENTACION DE HERRAMIENTAS CRIPTOGRAFICAS DE CLAVE PUBLICA Y FUNCIONES HASH PARA LA CUTB**", **p**ara su correspondiente evaluación

El desarrollo de dicho proyecto se llevo a cabo como requisito para obtener el título de **INGENIERO DE SISTEMAS**.

Agradeciendo de antemano su gentil colaboración y apoyo.

Cordialmente,

CARLOS SIERRA CASTILLO
Cod: 9305 133

JOSE CARLOS MONTES NARANJO
Cod:9305 550

ARTICULO 105.

La Corporación Universitaria Tecnológica de Bolívar, se reserva el derecho de propiedad intelectual de todos los trabajos de grado aprobados y no pueden ser explotados comercialmente sin su autorización.

Nota de aceptación

_____

_____

_____

_____
Presidente del Jurado

_____
Jurado

_____
Jurado

Fecha _____

A Dios, que me dio la fe y la fuerza
necesaria para seguir adelante.

A mi Mama Gladis por su paciencia y comprensión.

A mi Hijo Juan Camilo Por el tiempo que no te e podido dedicar.

A mi Esposa Alma por su incondicional apoyo.

Carlos

***A** Dios*

A mis padres, José y Vitélma, quienes en todo momento me brindaron su comprensión y apoyo, hasta llegar a la culminación de mi carrera.

José Carlos

AGRADECIMIENTOS

Los autores expresan sus agradecimientos a:

Jaime Arcila, Ingeniero Eléctrico y Director de este Trabajo de Grado.

Karol Medrano, por su constante apoyo y motivación.

Juan Manuel Montes, colaboradores en la parte Logística

La Corporación Universitaria Tecnológica de Bolívar.

Y todas aquellas personas que de una u otra forma colaboraron en la realización del presente trabajo.

INTRODUCCION

En este documento se encuentra plasmado la descripción de algoritmos criptográficos y funciones HASH propuestas en el trabajo de grado.

En el primer capítulo se incluyen todos los conceptos básicos y se introduce a los temas que se desarrollaran en los otros capítulos, además de brindar parte de la terminología empleada en el resto del documento. El segundo capítulo describe algunos Algoritmos de Criptografía Convencional, así como los algoritmos simétricos de cifrado como son el DES e IDEA. El tercer capítulo se estudian los algoritmos asimétricos o de clave pública RSA y LUC. En el cuarto capítulo se definen dos funciones HASH como son el MD5 y el SHA, y el Quinto, y último capítulo detalla las conclusiones y recomendaciones que los autores consideraron necesarias después de esta investigación.

Existen muchos algoritmos que no son descritos en este documento, puesto que no son del alcance de este proyecto, El núcleo de esta investigación es la implementación de los algoritmos propuestos, los cuales son mencionados en este documento, por eso para el entendimiento de los algoritmos es conveniente el conocimiento previo de la Teoría de Números.

Se ha pretendido que todos los conceptos queden suficientemente claros con la sola lectura de este libro, pero se recomienda vivamente que si el lector tiene interés por profundizar en cualquiera de los aspectos tratados aquí, consulte la bibliografía para ampliar sus conocimientos, pudiendo emplear como punto de partida las propias referencias que aparecen al final de este documento, aunque por desgracia, algunas de las más interesantes están en inglés.

TABLA DE CONTENIDO

CONCEPTOS BÁSICOS

ANEXO B

Referencia rápida.

Manual del Programador Invocación

De Librerías Criptográficas

ANEXO C
Manual de Usuario de WinCrip 2000
ANEXO D

Relación del Contenido del CD

LISTA DE FIGURAS

LISTA TABLAS

1. CONCEPTOS BASICOS

1.1 CRIPTOLOGIA

La* Criptología *es el estudio de los sistemas utilizados para las comunicaciones secretas. Esta rama del conocimiento envuelve a la* Criptografía *y al* Criptoanálisis*.

1.1.1 GRIPTOGRAFIA

Según el Diccionario de la Real Academia, la palabra Criptografía proviene del griego *Kriptos*, que significa oculto, y *Grafos* que significa escritura, y su definición es: "Arte de escribir con clave secreta o de un modo enigmático". Obviamente la Criptografía hace años que dejo de ser un arte para convertirse en una técnica, o mas bien un conglomerado de técnicas, que tratan sobre la protección, ocultamiento frente a observadores no autorizados, de la información. Entre las disciplinas que engloba cabe destacar la Teoría de Números o Matemáticas Discreta, que estudia las propiedades de los números enteros.

Existen dos documentos fundamentales, uno escrito por Claude Shannon en 1948 ("A Mathematical Theory of Communication"), en el que se sientan las bases de la Teoría de la Información, y que junto con otro articulo posterior del mismo autor sirvió de base para la Criptografía moderna. El segundo trabajo fundamental, publicado por Whitfield Diffie y Martin Hellman en

1976, se titulaba “New directions in Cryptography", e introducía el concepto de Criptografía de Clave Publica, abriendo enormemente el abanico de aplicación de esta disciplina.

Conviene hacer notar que la palabra Criptografía solo se refiere al uso de códigos, por lo que no engloba a las técnicas que se usan para romper dichos códigos (criptoanálisis). El termino Criptología, aunque no esta recogido aun en el Diccionario, se emplea habitualmente para agrupar estas dos disciplinas.

1.1.1.1 DEFINICION

La Criptografía, es la ciencia y estudio de lo que se escribe en secreto y de los diseños de los sistemas para dicho fin.

1.1.2 SISTEMA CRIPTOGRAFICO

Definiremos un criptosistema como una quíntupla (M; C; K; E; D***), donde:***

- ✓ M ***representa el conjunto de todos los mensajes sin cifrar (lo que se denomina texto plano) que pueden ser enviados.***
- ✓ *C* **representa el conjunto de todos los posibles mensajes cifrados, o criptogramas.**
- ✓ *K* **representa el conjunto de claves que se pueden emplear en el criptosistema.**
- ✓ *E* **es el conjunto de transformaciones de cifrado o familia de funciones que se aplica a cada elemento de M para obtener un elemento de** *C*. **Existe una transformación diferente** *Ek* **para cada valor posible de la clave** *k*.
- ✓ *D* **es el conjunto de transformaciones de descifrado, análogo a** *E*.

Figura 1 Modelo de un Sistema Criptografico

Todo criptosistema ha de cumplir la siguiente condición:

$$D_k(E_k(m)) = m$$

es decir, que si tenemos un mensaje m***, lo ciframos empleando la clave*** k ***y luego lo desciframos empleando la misma clave, obtenemos de nuevo el mensaje original*** m***.***

1.1.3 REQUERIMIENTOS

Un criptosistema debe cumplir con los siguientes requerimientos:

- **Las transformaciones de ciframiento y desciframiento deben ser eficientes para todas las claves.**
- **El sistema debe ser fácil de usar.**
- **La seguridad del sistema debe depender de las claves y no de los algoritmos de ciframiento y desciframiento.**

1.1.4 CLASIFICACION

Existen dos tipos fundamentales de criptosistemas:

1.1.4.1 SIMETRICOS

También conocidos como de *Clave Privada*. Son aquellos que emplean la misma clave k tanto para cifrar como para descifrar. Presentan el inconveniente de que para ser empleados en comunicaciones la clave *k* debe estar tanto en el emisor como en el receptor, lo cual nos lleva preguntarnos como transmitir la clave de forma segura.

1.1.4.2 ASIMETRICOS

También conocidos como de *Clave Publica*, que emplean una doble clave *(kp; kP)* . *kp* se conoce como clave privada y *kP* se conoce como clave publica. Una de ellas sirve para la transformación E de cifrado y la otra para la transformación *D* de descifrado. En muchos casos son intercambiables, esto es, si empleamos una para cifrar la otra sirve para descifrar y viceversa. Estos criptosistemas deben cumplir además que el conocimiento de la clave publica *kP* no permita calcular la clave privada *kp*. Ofrecen un abanico superior de posibilidades, pudiendo emplearse para establecer comunicaciones seguras por canales inseguros (puesto que únicamente viaja por el canal la clave publica, que solo sirve para cifrar), o para llevar a cabo autenticaciones.

1.1.5 CRIPTOANALISIS

El criptoanálisis consiste en comprometer la seguridad de un criptosistema. Esto se puede hacer descifrando un mensaje sin conocer la llave, o bien obteniendo a partir de uno o más criptogramas la clave que ha sido empleada en su codificación.

Uno de los tipos de análisis más interesante es el de texto plano escogido, que parte de que conocemos una serie de pares de textos planos elegidos por nosotros y sus criptogramas correspondientes. Esta situación se suele dar cuando tenemos acceso al dispositivo de cifrado y este nos permite efectuar operaciones, pero no nos permite leer su clave por ejemplo, las tarjetas de los teléfonos móviles. Él número de pares necesarios para obtener la clave desciende entonces significativamente. Cuando el sistema es débil, pueden ser suficientes unos cientos de mensajes para obtener información que permita deducir la

clave empleada. También podemos tratar de criptoanalizar un sistema aplicando el algoritmo de descifrado, con todas y cada una de las claves, a un mensaje codificado que poseemos y comprobar cuales de las salidas que se obtienen tienen sentido como posible texto plano.

***Este método y todos los que buscan exhaustivamente por el espacio de claves* K*, se denominan ataques por la* Fuerza Bruta.**

La Criptografía no solo se emplea para proteger información, también se utiliza para permitir su autenticación, es decir, para identificar al autor de un mensaje e impedir que nadie suplante su personalidad. En estos casos surge un nuevo tipo de criptoanálisis que esta encaminado únicamente a permitir que elementos falsos pasen por buenos. Puede que ni siquiera nos interese descifrar el mensaje original, sino simplemente poder sustituirlo por otro falso y que supere las pruebas de autenticación.

Como se puede apreciar, la gran variedad de sistemas criptográfico produce necesariamente gran variedad de técnicas de criptoanálisis, cada una de ellas adaptada a un algoritmo o familia de ellos. Con toda seguridad, cuando en el futuro aparezcan nuevos mecanismos de protección de la información, surgirán con ellos nuevos métodos de criptoanálisis

1.2 FUNCIONES HASH

1.2.1 DEFINICION

La función HASH es una función de la forma *h = H(M),* la cual es de una sola vía. Donde *M* es un Mensaje de Longitud Variable y *H* es el tipo de función HASH aplicada al mensaje, generando así una salida de tamaño fijo, proveyendo la capacidad de detección de errores.

Debido a que la función HASH no es considerada secreta, algunas veces es requerido proteger el valor HASH como se puede observar en las diversas formas de uso planteadas a continuación.

1.2.2 PROPIEDADES DE LA FUNCION HASH

Una función HASH *H* debe tener las siguientes propiedades:

I. ***H* puede ser aplicado a bloques de cualquier tamaño.**
II. ***H* produce una salida de longitud fija.**
III. ***H(x)* es relativamente fácil de calcular para cualquier *x* dada; ya sean implementaciones en hardware o en software.**
IV. **Para cualquier código *M* es computacionalmente no factible encontrar *x* tal que *H(x)=M.***
V. **Para cualquier bloque *x* dado es computacionalmente no factible encontrar *y* [1] *x* tal que *H(y)=H(x).***
VI. **Es computacionalmente no factible encontrar cualquier par *(x, y)* tal que *H(x)=H(y).***

1.2.3 FORMAS DE USO DE LAS FUNCIONES HASH

Esta función puede ser usada de las siguientes formas:

- **El mensaje junto con el código HASH concatenado son Encriptados usando un esquema de criptografía convencional. En este esquema se provee autenticación.**

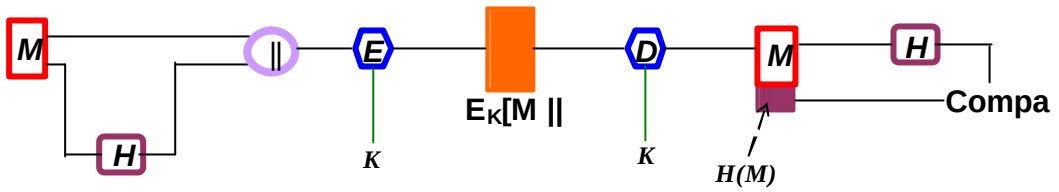


Figura 2 Primer Esquema de Uso de la Función HASH

- **El mensaje es concatenado con el código HASH Encriptados usando un esquema de criptografía convencional. Se provee autenticación.**

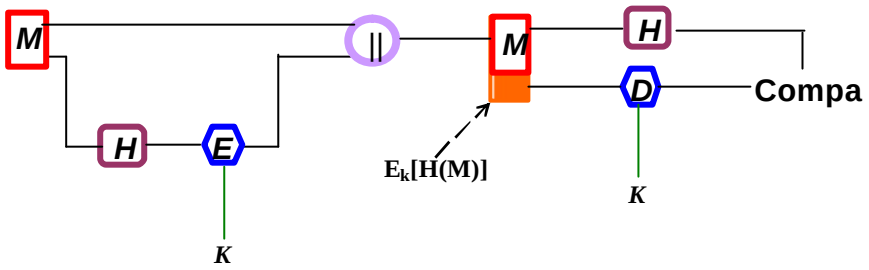


Figura 3 Segundo Esquema de Uso de la Función HASH

- **El código HASH es Encriptado usando un esquema de criptografía de clave pública y usando la clave privada del emisor. Se provee autenticación y firma digital.**

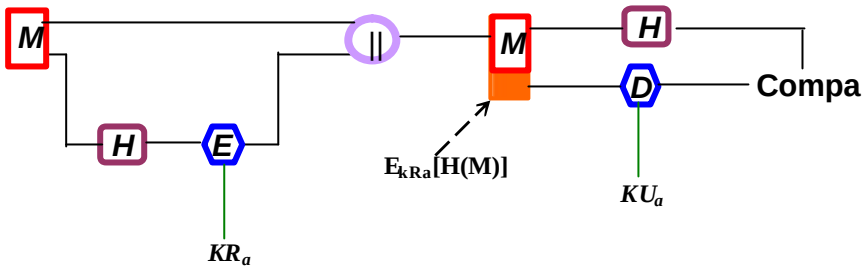


Figura 4 Tercer Esquema de Uso de la Función HASH

- **Un esquema mixto donde se usa la criptografía convencional y la de clave pública, con el fin de proveer los servicios de confidencialidad, autenticación y firma digital. En este caso el código HASH es Encriptado usando un esquema de criptografía de clave pública usando la clave privada del emisor y finalmente todo el bloque resultante es Encriptado usando la criptografía convencional.**

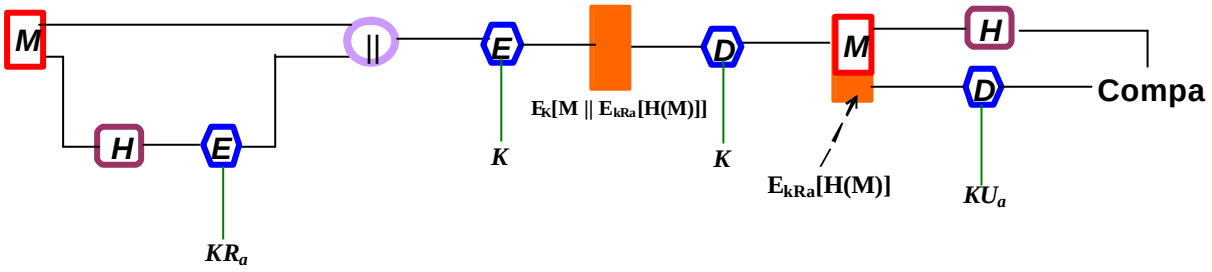


Figura 5 Cuarto Esquema de Uso de la Función HASH

- **Una técnica que provee autenticación pero sin Encripción: Dos partes comparten un valor secreto *S* el cual es concatenado con *M* y después es aplicada la función HASH.**

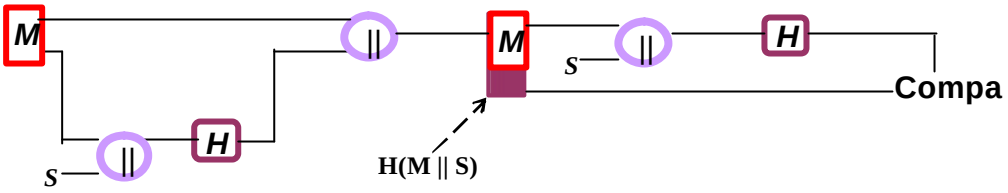


Figura 6 Quinto Esquema de Uso de la Función HASH

- **Una variante al método anterior para agregar confidencialidad encriptando el mensaje junto con el código HASH usando la criptografía convencional.**

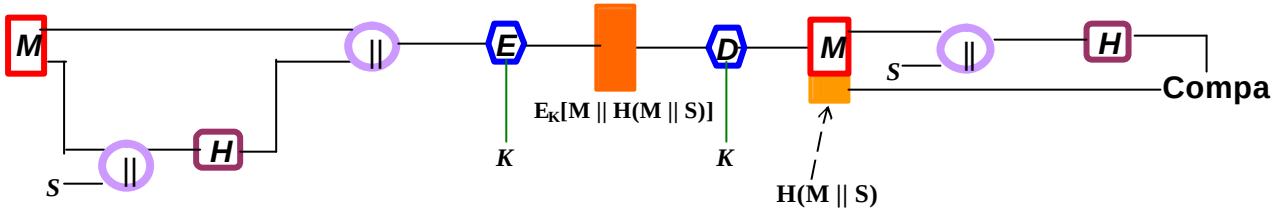


Figura 7 Sexto Esquema de Uso de la Función HASH

2. ALGORITMOS DE CRIPTOGRAFIA CONVENCIONAL

2.1 CRIPTOGRAFIA CONVENCIONAL

El ser humano siempre ha tenido secretos de muy diversa índole, y ha buscado mecanismos para mantenerlos fuera del alcance de miradas indiscretas.* Julio Cesar *empleaba un sencillo algoritmo para evitar que sus comunicaciones militares fueran interceptadas.* Leonardo Da Vinci *escribía las anotaciones sobre sus trabajos de derecha a izquierda y con la mano zurda. Otros personajes, como* Sir Francis Bacon *o* Edgar Allan Poe *eran conocidos por su afición a los códigos criptográficos, que en muchas ocasiones constituían un apasionante divertimento y un reto para el ingenio.

Todos los algoritmos criptográficos clásicos son simétricos, ya que hasta mediados de los

años setenta no nació la Criptografía asimétrica..

2.2 ALGORITMOS PARA CIFRADORES DE SUSTITUCION

Se engloban dentro de este apartado todos los algoritmos criptográficos que, sin desordenar los símbolos dentro del mensaje, establecen una correspondencia única para todos ellos en todo el texto. Es decir, si al

símbolo *A* le corresponde el símbolo *D*, esta correspondencia se mantiene a lo largo de todo el mensaje.

2.2.1 JULIO CESAR

2.2.1.1 DESCRIPCION

Este es el cifrador más sencillo de sustitución el cual fue inventado por Julio Cesar. Consiste en reemplazar cada letra del alfabeto por otra letra que se mueve k lugares en el alfabeto, donde k es la clave.

Si las letras del alfabeto desde la A hasta la Z, se numeran de 1 a 27

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

Las funciones de Encripción y Desencripción son las siguientes:

$$E_k(M,k) = \forall i \in \mathrm{Z}(1 \le i \le |M|)\left[C_i = M_i + k \mod 27\right]$$

$$D_{k'}(C,k') = \forall i \in \mathrm{Z}(1 \le i \le |C|)\left[M_i = C_i - k' \mod 27\right]$$

$$k' = k$$

Ejemplo : k = 3

A—>D, B—>E, C—>F, ..., X—>A, Y—>B, Z—>C.

Mensaje : M E N S A J E E N V I A D O A Y E R

Criptograma : P H Q V D M H H Q Y L D G R D B H U

2.2.1.2 CRIPTOANALISIS

Este cifrador no es resistente a un ataque de fuerza bruta, el cual es muy fácil de realizar con el menor esfuerzo computacional, puesto que solo hay 26 posibles claves (En el Idioma español).

2.2.1.3 IMPLEMENTACION

Se puede realizar de muchas maneras, una de ellas es mediante una función, la cual recibe los siguientes parámetros:

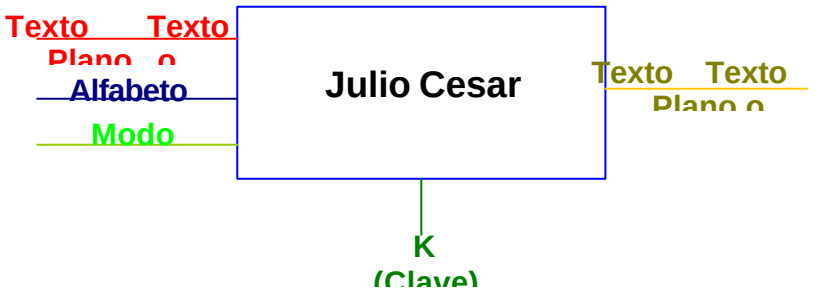


Figura 8 Implementación Cifrador Julio Cesar

Texto Plano o Texto Cifrado, **dependiendo del *modo* en que se encuentre se introduce ya sea al texto plano o el cifrado, mediante una cadena o un archivo especificando su nombre, la salida de la función también depende del modo como se puede notar en la gráfica.**

Alfabeto, **contienen los valores entre los cuales va oscilar el alfabeto que va a utilizar la función o algoritmo. El orden que indican este parámetro se utiliza para calcular el valor de cada carácter que contiene el archivo o cadena de entrada este orden es el mismo que tienen los caracteres del código ASCII. El máximo valor de alfabeto que se puede usar es 256 el cual es el máximo valor del código ASCII.**

Modo, **indica si el algoritmo se utiliza ya se para Encriptar (Modo=0) o Desencriptar (Modo<>1).**

K o Clave, **la cual se vale este algoritmo para poder Encriptar o Desencriptar el archivo de entrada. Su valor máximo depende de tamaño del alfabeto**

2.2.2 MONOALFABETICO

2.2.2.1 DESCRIPCION

Este cifrador utiliza una sustitución arbitraria o en otras palabras la clave es una permutación de 27 caracteres alfabéticos.

Las funciones de Encripción y Desencripción son las siguientes:

$$E_k(M,k) = \forall i \in \mathrm{Z}(1 \le i \le |M|)\left[C_i = P_k(M_i)\right]$$

$$D_{k'}(C,k') = \forall i \in \mathrm{Z}(1 \le i \le |C|)\left[M_i = P'_{k'}(C_i)\right]$$

$$k' = k$$

Donde, $P_k(\Sigma) = \Sigma$ **, es una función cuyo dominio y rango son las letras del alfabeto y está dada por una permutación de estas letras del alfabeto que depende de** k **. Igualmente,** $P'_{k'}(\Sigma) = \Sigma$ **, es una función cuyo dominio y rango también son las letras del alfabeto y está dada por una permutación de las letras del alfabeto que depende de** k' **.**

Como se verá estas funciones están relacionadas porque son inversas entre sí:

Ejemplo :

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

Mensaje : C R I P T O G R A F I A

Criptograma : H Z M D W F I Z G Ñ M G

2.2.2.2 CRIPTOANALISIS

El cifrado monoalfabetico constituye la familia de métodos más simple de criptoanalizar, puesto que las propiedades estadísticas del texto plano se conservan en el Criptograma. Supongamos que, por ejemplo, la letra que más aparece en Castellano es la *A*. Parece lógico que la letra más frecuente en el texto codificado sea aquella que corresponde con la *A*. Emparejando las frecuencias relativas de aparición de cada símbolo en el mensaje cifrado con el histograma de frecuencias del idioma en el que se supone esta el texto plano, podremos averiguar fácilmente la clave.

2.2.2.3 IMPLEMENTACION

Se puede realizar de muchas maneras, una de ellas es mediante una función, la cual recibe los siguientes parámetros:

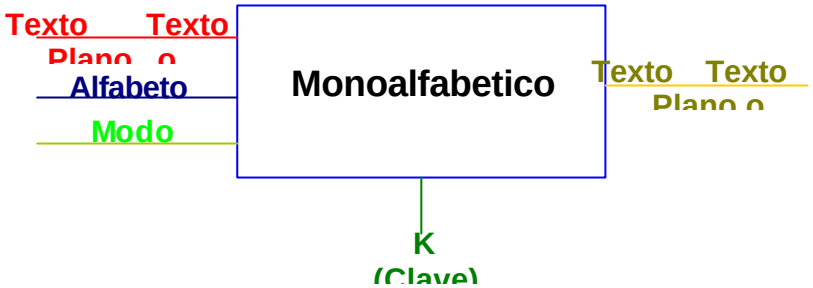


Figura 9 Implementación Cifrador Monoalfabetico

Texto Plano o Texto Cifrado, **dependiendo del *modo* en que se encuentre se introduce ya sea al texto plano o el cifrado, mediante una cadena o un archivo especificando su nombre, la salida de la función también depende del modo, como se puede notar en la gráfica.**

Alfabeto, **contienen los valores entre los cuales va oscilar el alfabeto que va a utilizar la función o algoritmo. El orden que indican este parámetro se utiliza para calcular el valor de cada carácter que contiene el archivo o cadena de entrada este orden es el mismo que tienen los caracteres del código ASCII. El máximo valor de alfabeto que se puede usar es 256 el cual es el máximo valor del código ASCII.**

Modo, **indica si el algoritmo se utiliza ya se para Encriptar (Modo=0) o Desencriptar (Modo<>1).**

K o Clave, **de la cual se vale esta función o algoritmo para poder Encriptar o Desencriptar el archivo o cadena de entrada. Este parámetro es un vector el cual debe contener una permutación de los elementos especificados en el alfabeto; el tamaño máximo de este vector es 255 y esto se da cuando el usuario escoge como alfabeto todo los caracteres del código ASCII; de lo contrario su tamaño deberá ser igual al numero de elementos del alfabeto escogido.**

2.2.3 VIGENERE

2.2.3.1 DESCRIPCION

Este cifrador de sustitución está clasificado dentro de los polialfabéticos[1], debido a que un conjunto de reglas de sustitución monoalfabéticas son usadas y la clave es la que determina que reglas en particular se debe escoger para cada transformación.

[1] En los cifrados poli alfabéticos la sustitución aplicada a cada carácter varia en función de la posición que ocupe este dentro del texto plano. En realidad corresponde a la aplicación cíclica *de n* cifrados monoalfabeticos.

El cifrador Vigenère está basado en la tabla que se muestra a continuación, correspondiente al conjunto de sustituciones del cifrador de Julio Cesar.

Tabla 1 *Sustitución de Alfabetos (Tabla Vigenère)*

	a	b	c	d	e	f	g	h	i	j	K	l	m	n	ñ	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**	**Ñ**	**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**	**Ñ**	**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**	**Ñ**	**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**	**Ñ**	**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**	**Ñ**	**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**	**Ñ**	**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**	**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**	**Ñ**	**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**	**Ñ**	**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**	**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**	**Ñ**	**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**	**Ñ**	**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**	**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**	**Ñ**	**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**
ñ	**Ñ**	**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**
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**	**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**	**Ñ**	**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**	**Ñ**	**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**	**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**	**Ñ**	**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**	**Ñ**	**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**	**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**	**Ñ**	**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**	**Ñ**	**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**	**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**	**Ñ**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**
z	**Z**	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**Ñ**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**

Las funciones de Encripción y Desencripción son las siguientes:

$$E_k(M,k) = \forall i \in Z(1 \le i \le |M|)\left[C_i = M_i + k_{i \bmod |k|} \bmod 27\right]$$

$$D_{k'}(C,k') = \forall i \in Z(1 \le i \le |C|)\left[M_i = C_i - k'_{i \bmod |k'|} \bmod 27\right]$$

$$k' = k$$

Ejemplo **:** ***Clave:*** **CASAS**

Mensaje :	**C**	**R**	**I**	**P**	**T**	**O**	**G**	**R**	**A**	**F**	**I**	**A**
Clave :	**C**	**A**	**S**	**A**	**S**	**C**	**A**	**S**	**A**	**S**	**C**	**A**
Criptograma:	**E**	**R**	**A**	**P**	**M**	**Q**	**G**	**K**	**A**	**X**	**K**	**A**

2.2.3.2 CRIPTOANALISIS

Para criptoanalizar este tipo de claves basta con efectuar d análisis estadísticos independientes agrupando los símbolos según la *ki* empleada para codificarlos. Para estimar |*k*|, buscaremos la periodicidad de los patrones comunes que puedan aparecer en el texto cifrado. Obviamente, para el criptoanálisis, necesitaremos al menos |*k*| veces mas cantidad de texto que con los métodos monoalfabeticos.

2.2.3.3 IMPLEMENTACION

Se puede realizar de muchas maneras, una de ellas es mediante una función, la cual recibe los siguientes parámetros:

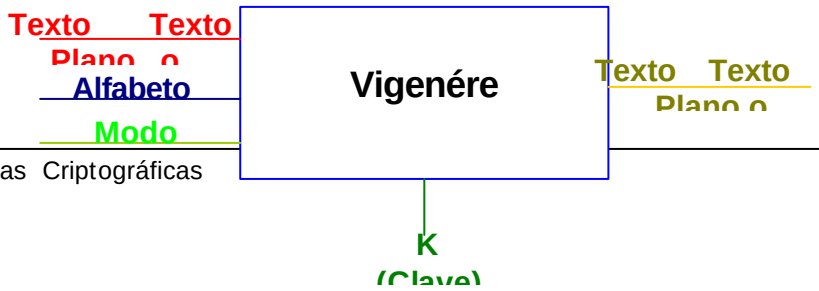


Figura 10. Implementación Cifrador Vigenère

Texto Plano o Texto Cifrado, **dependiendo del *modo* en que se encuentre se introduce ya sea al texto plano o el cifrado, mediante una cadena o un archivo especificando su nombre, la salida de la función también depende del modo, como se puede notar en la gráfica.**

Alfabeto, **contienen los valores entre los cuales va oscilar el alfabeto que va a utilizar la función o algoritmo. El orden que indican este parámetro se utiliza para calcular el valor de cada carácter que contiene el archivo o cadena de entrada este orden es el mismo que tienen los caracteres del código ASCII. El máximo valor de alfabeto que se puede usar es 256 el cual es el máximo valor del código ASCII.**

Modo, **indica si el algoritmo se utiliza ya se para Encriptar (Modo=0) o Desencriptar (Modo<>1).**

K o Clave, **la cual se vale este algoritmo o función para poder Encriptar o Desencriptar el archivo de entrada. Los elementos que conforman la clave deben estar dentro del alfabeto especificado la cual se vale este algoritmo para poder Encriptar o Desencriptar el archivo de entrada. Su valor máximo depende de tamaño del alfabeto**

2.3 ALGORITMOS PARA CIFRADORES DE PRODUCTO

La gran mayoría de los algoritmos de cifrado simétricos se apoyan en los conceptos de confusión y difusión, que se combinan para dar lugar a los denominados cifrados de producto.

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

2.3.1 DES

El método criptográfico representativo de los criptosistemas simétricos es el denominado "Data Encryption Standard' (DES) adoptado en 1977 por National Bureau of Standards, ahora National Institute of Standards and Technology (NIST) como un estándar de procesamiento de información federal. El método DES es un cifrador por producto que combina diversas sustituciones y transposiciones basadas en el manejo de determinadas permutaciones.

Los pasos de los cuales se conforma el DES, convierten a dicho método en uno de los más eficaces contra el ataque de los criptoanalistas. Para esto, cada uno de estos pasos fueron desarrollados en base a diversos fundamentos matemáticos que constituyen el pilar de la fortaleza del método.

2.3.1.1 DESCRIPCION

El método de cifrado DES, es considerado un sistema de bloques cifradores, posee tres elementos básicos en su proceso: el texto plano, el cual es un bloque de 64 bits, una clave criptográfica cuya longitud es de 56 bits y un Criptograma, que igual que el texto plano posee una longitud de 64 bits. La longitud de cada uno de setos elementos fue seleccionada con el objeto de reducir las posibilidades de que el método pueda ser roto ante el ataque de criptoanálisis. Las bases matemáticas del proceso, involucran el manejo de tres áreas principales: álgebra abstracta y álgebra lineal.

El proceso de cifrado por medio de este método consta de 3 pasos: el primer paso utiliza una permutación inicial IP, la cual es un bloque cifrador construido a partir de cruce de líneas; el segundo paso consta de 16 iteraciones, en las cuales se utilizan diversas permutaciones y especialmente las cajas-s, cuya estructura provee la razón de la fortaleza del DES; finalmente, el tercer paso utiliza la inversa de la permutación inicial , es decir IP^{-1}. Cada uno de los pasos y elementos usados en la Encripción DES posee su respectivo fundamento matemático, pero el criterio utilizado para el diseño de las permutaciones y de las cajas-s correspondientes, esta sujeta a muchas especulaciones, pues los diseñadores del método no están dispuestos a revelar el porque de las estructuras de estos elementos.

2.3.1.1.1 ENCRIPCION

El esquema general de Encripción del DES se ilustra en la siguiente Figura en donde se puede ver que hay dos entradas: El texto plano formado por una cadena de 64 bits de longitud y la clave formada por una cadena de 56 bits de longitud.

El algoritmo de Encripción se divide en tres fases, tal como se puede observar en la columna más izquierda *Figura 11*

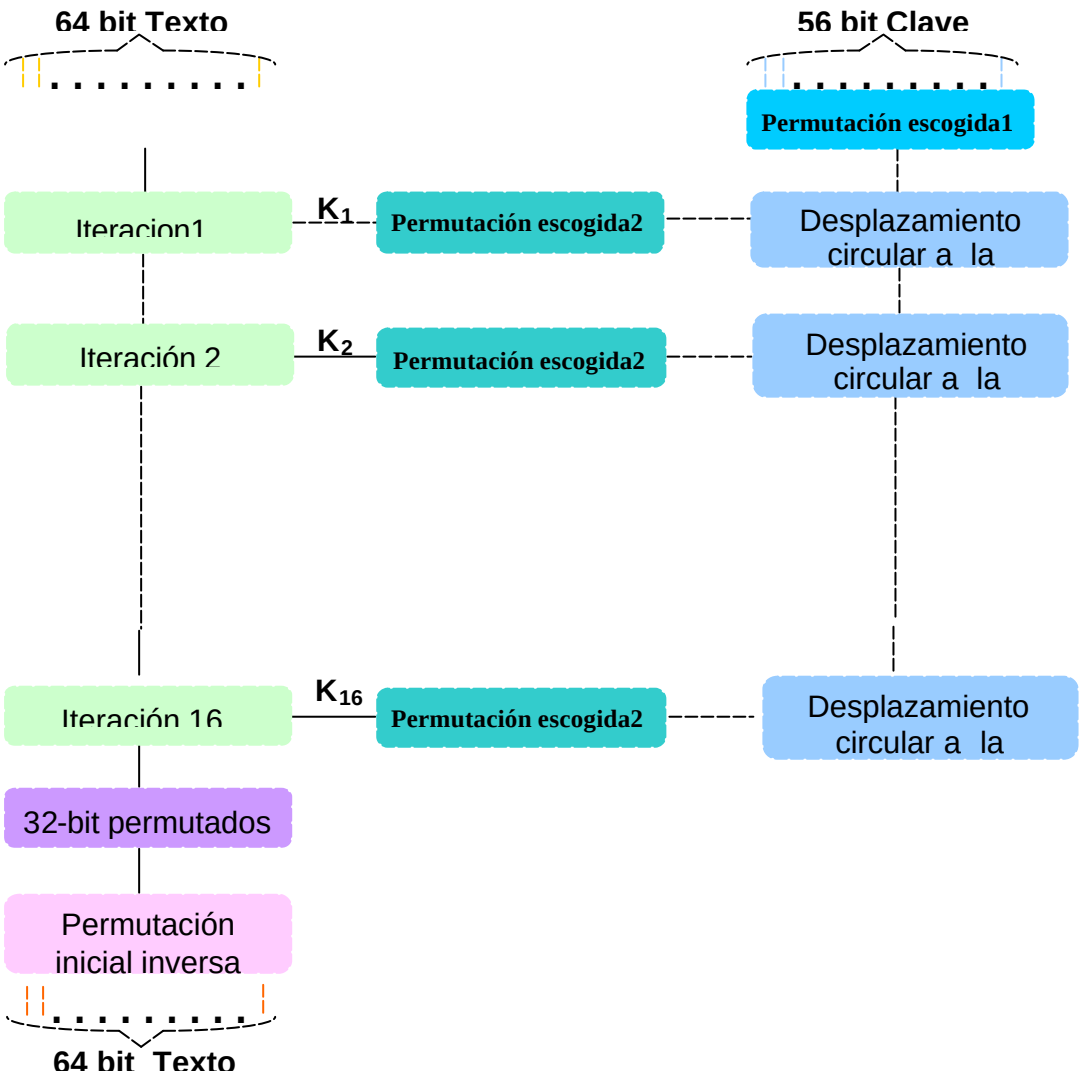


Figura 11 Descripción General Del Cifrador DES

- ***Fase 1*: Los 64 bits del texto plano pasan por una permutación inicial donde se reordenan los bits para producir una entrada permutada.**

- ***Fase 2:* Consiste en 16 iteraciones de la misma función, la cual tiene funciones internas de permutaciones y sustituciones.**

- ***Fase 3*: Recibe un bloque de 64 bits el cual es función del bloque inicial y de la clave. Este bloque es dividido en dos bloques de 32 bits, los cuales son intercambiados para producir una pre-salida. Finalmente la pre-salida es pasada por una permutación que es inversa a la permutación inicial, para producir un Criptograma de 64 bits.**

La parte derecha de la *Figura 11* muestra la forma como la clave de 56 bits es usada. Inicialmente la clave pasa por una función de permutación y después por cada una de las 16 iteraciones una subclave k_J es generada usando la combinación de un desplazamiento circular y una permutación.

2.3.1.1.1.1 PERMUTACION INICIAL Y SU INVERSA

En el método DES se utiliza una permutación inicial IP y una permutación final IP^{-1}. Se puede demostrar que la una es la inversa de la otra de la siguiente manera: considere los 64 bits de entrada M.

M_1 M_2 M_3 M_4 M_5 M_6 M_7 M_8 M_9 M_{10} M_{11} M_{12} M_{13} M_{14} M_{15} M_{16}

M_{17} M_{18} M_{19} M_{20} M_{21} M_{22} M_{23} M_{24} M_{25} M_{26} M_{27} M_{28} M_{29} M_{30} M_{31} M_{32}
M_{33} M_{34} M_{35} M_{36} M_{37} M_{38} M_{39} M_{40} M_{41} M_{42} M_{43} M_{44} M_{45} M_{46} M_{47} M_{48}
M_{49} M_{50} M_{51} M_{52} M_{53} M_{54} M_{55} M_{56} M_{57} M_{58} M_{59} M_{60} M_{61} M_{62} M_{63} M_{64}

Donde M_i es un dígito binario. Entonces la permutación X = IP(M) es como sigue:

M_{58} M_{50} M_{42} M_{34} M_{26} M_{18} M_{10} M_2 M_{60} M_{52} M_{44} M_{36} M_{28} M_{20} M_{12} M_4
M_{62} M_{54} M_{46} M_{38} M_{30} M_{22} M_{14} M_6 M_{64} M_{56} M_{48} M_{40} M_{32} M_{24} M_{16} M_8
M_{57} M_{49} M_{41} M_{33} M_{25} M_{17} M_9 M_1 M_{59} M_{51} M_{43} M_{35} M_{27} M_{19} M_{11} M_3
M_{61} M_{53} M_{45} M_{37} M_{29} M_{21} M_{13} M_5 M_{63} M_{55} M_{47} M_{39} M_{31} M_{23} M_{15} M_7

Si nosotros tomamos la permutación inversa $Y = IP^{-1}(X) = IP^{-1}(IP(M))$, el orden de los bits originales serán restaurados. Además de la IP y su inversa, encontramos también las permutaciones E, P, PC- 1 y PC-2 que más adelante veremos.

Las permutaciones se utilizan con el objetivo de aplicar la técnica criptográfica denominada *difusión.* Estas permutaciones son transformaciones lineales sobre $Z_{2,n}$ utilizadas para Encriptar el texto plano. La longitud de la entrada a una permutación puede ser de un tamaño cualquiera.

El DES es considerado un sistema de bloques cifradores, y cada una de las permutaciones utilizadas en el proceso son simplemente un bloque cifrador. Estos bloques se diseñaron siguiendo el método de *cruce de líneas.* Un cruce de líneas es una primitiva utilizada para construir bloques cifradores y se basa en las siguiente esquema:

Si se tiene

τ : **una permutación de enteros** $(0,1,.....,N-1)$

π_τ : **$(x_0, x_1,.....,x_{N-1})$ ® $(x_{t(0)}, x_{t(1)},.....,x_{t(N-1)})$**

Un cruce de líneas es un caso especial de una transformación lineal no singular. La i-esima fila de una matriz L.

$$\begin{matrix} 1_{0,0} & 1_{0,1} & ..... & 1_{0,N-1} \\ 1_{1,0} & 1_{1,1} & ..... & 1_{1,N-1} \\ 1_{N-1,0} & 1_{N-1,1} & ..... & 1_{N-1,N-1} \end{matrix}$$

Corresponde a p_t con un 1 en la columna j, donde j = t(i).

Cabe destacar que no se menciona el criterio de selección de las permutaciones, debido a que los diseñadores y desarrolladores del método no han revelado aun la naturaleza de estas, sin embargo, es posible que se hayan escogido con el fin de poder poner en práctica de la mejor manera la difusión en la encriptación.

Las permutaciones inicial e inversa son definidas mediante las *Tablas 2 y 3* respectivamente.

TABLA 2 Permutación Inicial (IP**)**

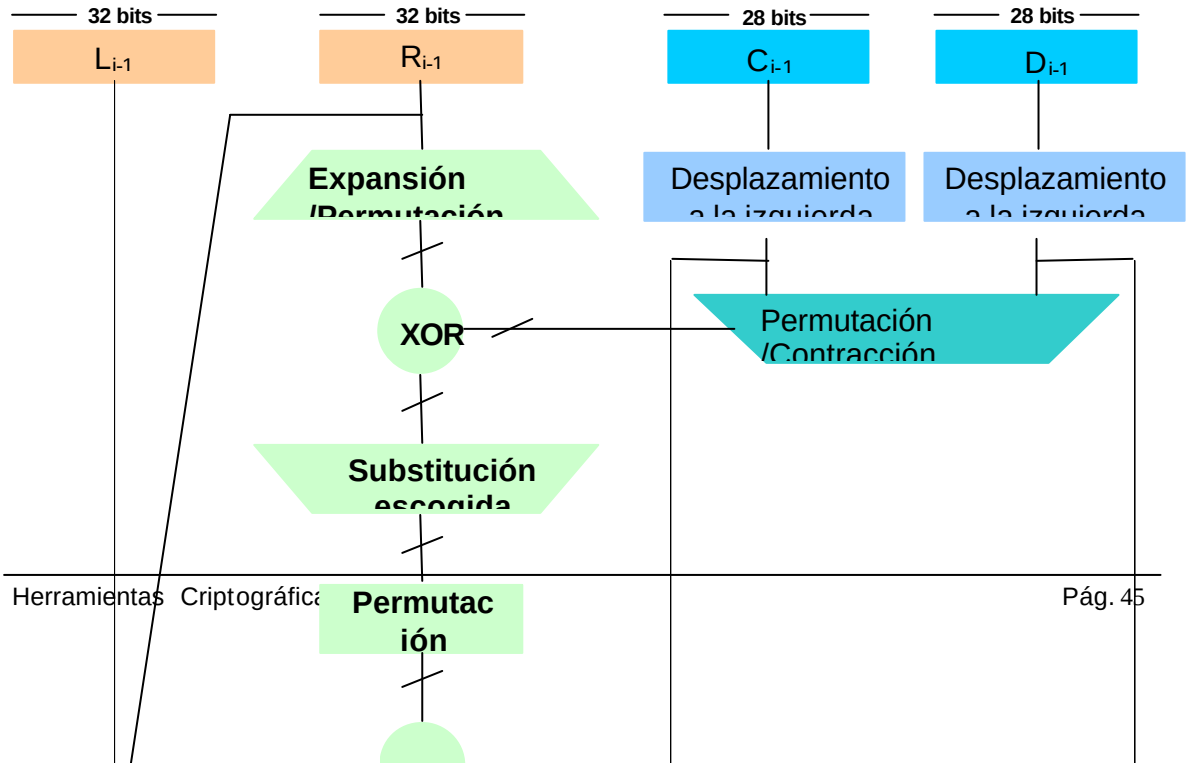
Bit de Salida	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Desde Bit de entrada	58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
Bit de Salida	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Desde Bit de entrada	62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8
Bit de Salida	33	34	35	36	37	38	39	40	41	42	43	44	45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

TABLA 3 Permutación Inicial Inversa (IP^{-1})

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

2.3.1.1.1.2 ITERACIONES

En la *Figura 12* se muestra la forma como se realiza cada una de las Iteraciones del Algoritmo de Encripción del DES. En esencia, el bloque de 64 bits permutado de la entrada pasa a través de las 16 iteraciones, produciendo un valor intermedio de 64 bits a la conclusión de cada iteración. Ahora analizando lo que hace cada iteración, observando inicialmente el sector izquierdo de la *Figura 12*, donde se puede ver que el bloque de 64 bits que recibe cada iteración es tratado por separado por dos bloques de 32 bits cada uno(Uno izquierdo (L) y otro derecho (R)).



Las partes izquierda y derecha de cada valor intermedio de 64 bits, son tratadas como cantidades separadas de 32 bits, etiquetadas como L y R. El procesamiento general, en cada iteración, puede ser resumido por las siguientes formulas:

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \text{ Å } f(R_{i-1}, K_i)$$

La función f es ilustrada en la siguiente *Figura 13*:

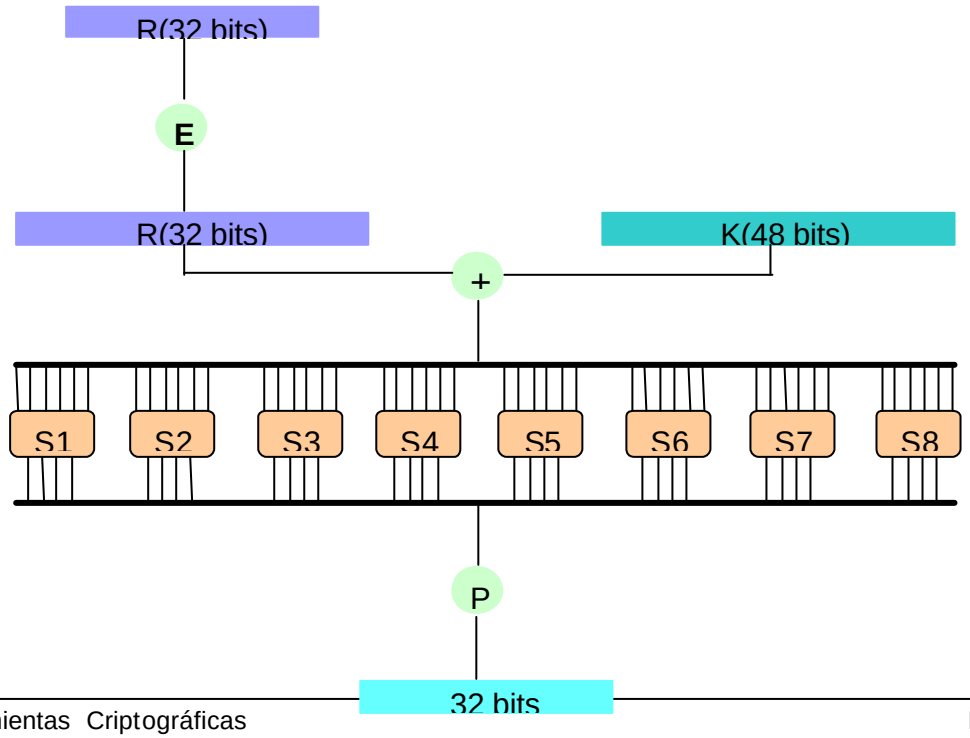


Figura 13 Calculo De f(R,K)

Los 32 bits del bloque derecho(R_{i-1}) pasan por una función de Permutación - Expansión (E) generando un bloque de 48 bits de acuerdo a la *Tabla 4*.

TABLA 4 Permutación - Expansión (E)

Bit de Salida	1	2	3	4	5	6	7	8	9	10	11	12
Desde Bit de entrada	32	1	2	3	4	5	4	5	6	7	8	9
Bit de Salida	13	14	15	16	17	18	19	20	21	22	23	24
Desde Bit de entrada	8	9	10	11	12	13	12	13	14	15	16	17
Bit de Salida	25	26	27	28	29	30	31	32	33	34	35	36
Desde Bit de entrada	16	17	18	19	20	21	20	21	22	23	24	25
Bit de Salida	37	38	39	40	41	42	43	44	45	46	47	48
Desde Bit de entrada	24	25	26	27	28	29	28	29	30	31	32	1

A partir del resultado obtenido de 48 bits junto con la clave k_i se obtiene un nuevo bloque de 48 bits mediante la operación XOR (O exclusivo) entre estos últimos.

Este nuevo bloque de 48 bits es pasado por una función de sustitución que produce una salida de 32 bits, el cual es permutado con una función de permutación (P) tal como se define en la *Tabla 5.*

TABLA 5. Función de Permutación (P)

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

2.3.1.1.1.3 CAJAS-S

Las cajas-s se usan con el objetivo de aplicar la técnica criptográfica denominada *confusión* y se basan en transformaciones de sustitución. Cada sustitución es una transformación no lineal realizada para bloques de pequeños tamaños. Las sustituciones logran que la salida no sea una función lineal de la entrada.

Al igual que con las permutaciones, las cajas-s poseen su naturaleza oculta, pues tampoco para estas se han rebelado los criterios utilizados para su selección. Al parecer las cajas-s poseen dentro de su estructura propiedades especiales o trampas ocultas que constituyen la verdadera fortaleza del método. También es posible que esta gran fortaleza pueda convertirse en la gran debilidad del método si dichas trampas y propiedades sean descubiertas.

A pesar del gran misterio alrededor de las cajas-s, se pueden citar 3 propiedades que los diseñadores han publicado. Estas son:

- **Las cajas-s no son una función lineal de la entrada.**
- **Cambiando un bit en la entrada de un cajas-s resulta un cambio de al menos dos bits en la salida.**
- Las cajas-s fueron seleccionadas con el fin de minimizar la diferencia entre el número de 1s y 0s cuando una simple entrada permanece constante.

La función de sustitución(confusión) consiste de un conjunto de 8 cajas S, cada una de las cuales reciben 6 bits como entrada y produce 4 bits como salida.

Estas transformaciones se definen en la *Tabla 6* la cual se interpreta de la siguiente manera:

- ✓ **El primero y último bits de los seis de entrada forman un número binario de 2 bits que sirve para seleccionar una fila particular en la tabla para S_i.**
- ✓ **Los cuatro (4) bits de la mitad forman un número binario de 4 bits que sirve para seleccionar una columna particular en la tabla para S_i.**
- ✓ **El número decimal obtenido en la *Tabla 6* de acuerdo a la fila y columna es convertido a binario y cuya representación serán los correspondientes 4 bits de salida.**

TABLA 6 Definición de Cajas S_i del DES

	Número de columnas																
Filas	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Cajas**
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7	
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8	S_1
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0	
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13	
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10	
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5	S_2
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15	
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9	
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8	
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1	S_3
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7	
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12	
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15	
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9	S_4
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4	
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14	
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9	
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6	S_5
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14	
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3	
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11	
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8	S_6
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6	
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13	
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1	
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6	S_7
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2	
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12	
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7	
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2	S_8
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8	
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11	

2.3.1.1.1.4 GENERACION DE CLAVES

En la parte derecha de la figura del Algoritmo General de Encripción del DES, se puede ver que la clave de entrada de 56 bits pasa inicialmente por una permutación establecida mediante la *Tabla 7*, de donde se generan 56 bits permutados.

TABLA 7 Permutación escogida 1 (PC-1)

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

El bloque de 56 bits obtenido es ahora tratado en dos partes de 28 bits cada una C_0 y D_0, desplazándonos ahora a la parte derecha de la *Figura 12* de la Simple Iteración del Algoritmo DES.
Para cada iteración, independientemente tanto a C_i como a D_i se les aplica un desplazamiento circular a la izquierda de 1 o 2 bits de acuerdo a la *Tabla 8.*

TABLA 8 Desplazamientos a la izquierda

Número Iteración	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bits rotados	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Finalmente los dos bloques obtenidos se utilizarán en la siguiente iteración formando C_{i+1} y D_{i+1}, y por otra parte se unen para formar un bloque de 56 bits, los cuales serán la entrada de una nueva función de permutación-

escogencia que produce la clave de salida de la iteración k_i de acuerdo a la *Tabla 9*.

TABLA 9 Permutación escogida 2 (PC-2)

Bit de Salida	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Desde Bit de entra	14	17	11	24	1	5	3	28	15	6	21	10	23	19	12	4
Bit de Salida	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Desde Bit de entra	26	8	16	7	27	20	13	2	41	52	31	37	47	55	30	40
Bit de Salida	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
Desde Bit de entra	51	45	33	48	44	49	39	56	34	53	46	42	50	36	29	32

2.3.1.1.2 DESENCRIPCION

El proceso es básicamente el mismo de encripción con la única diferencia que las claves se usan en sentido inverso desde k_{16} hasta k_1, tal como se muestra en la siguiente *Figura 14*:

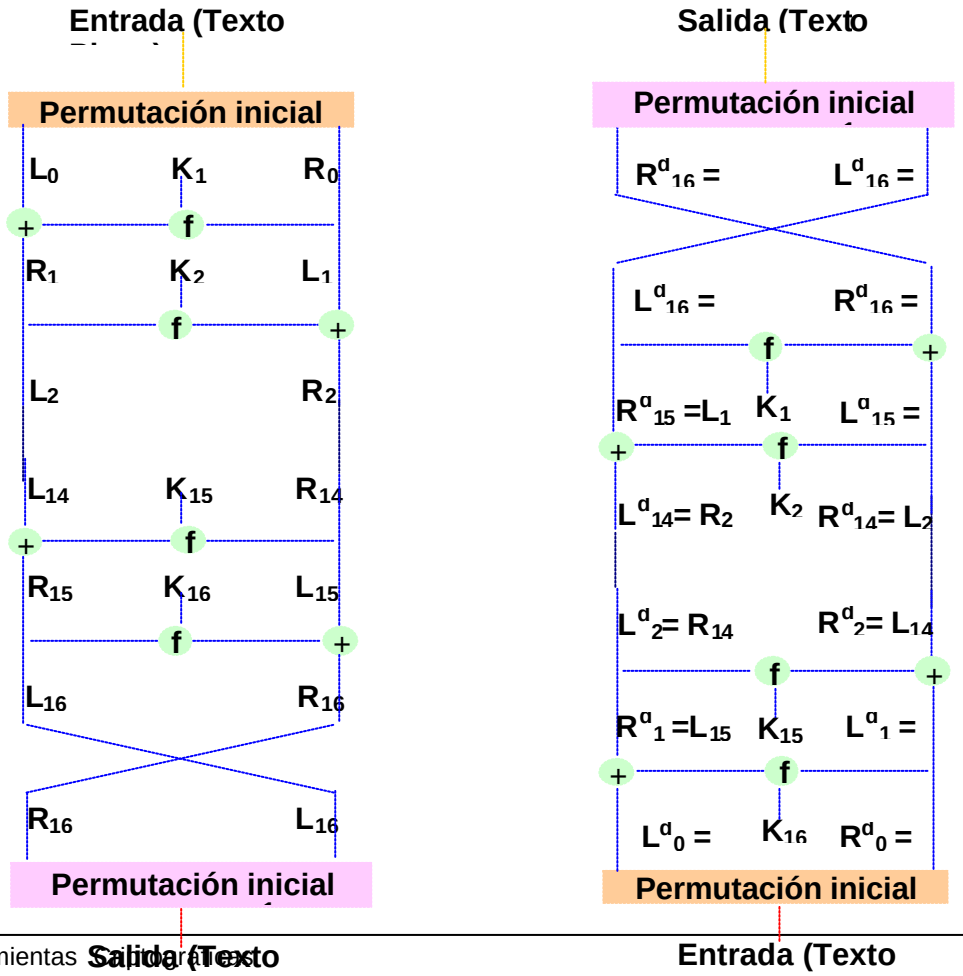


Figura 14 Encripción y Desencripción Del DES

2.3.1.1.3 MODOS DE OPERACIÓN

Modo electronic codebook (ECB). **El texto plano es manejado 64 bits a la vez, y cada bloque de texto plano es encriptado usando la misma clave. Para un mensaje mayor de 64 bits, se parte el mensaje en bloques de 64 bits, rellenando el último bloque si es necesario. La desencripción es ejecutada un bloque a la vez, siempre usando la misma clave. El método es ideal para encriptar datos de tamaño pequeño, como una clave de encriptación.**

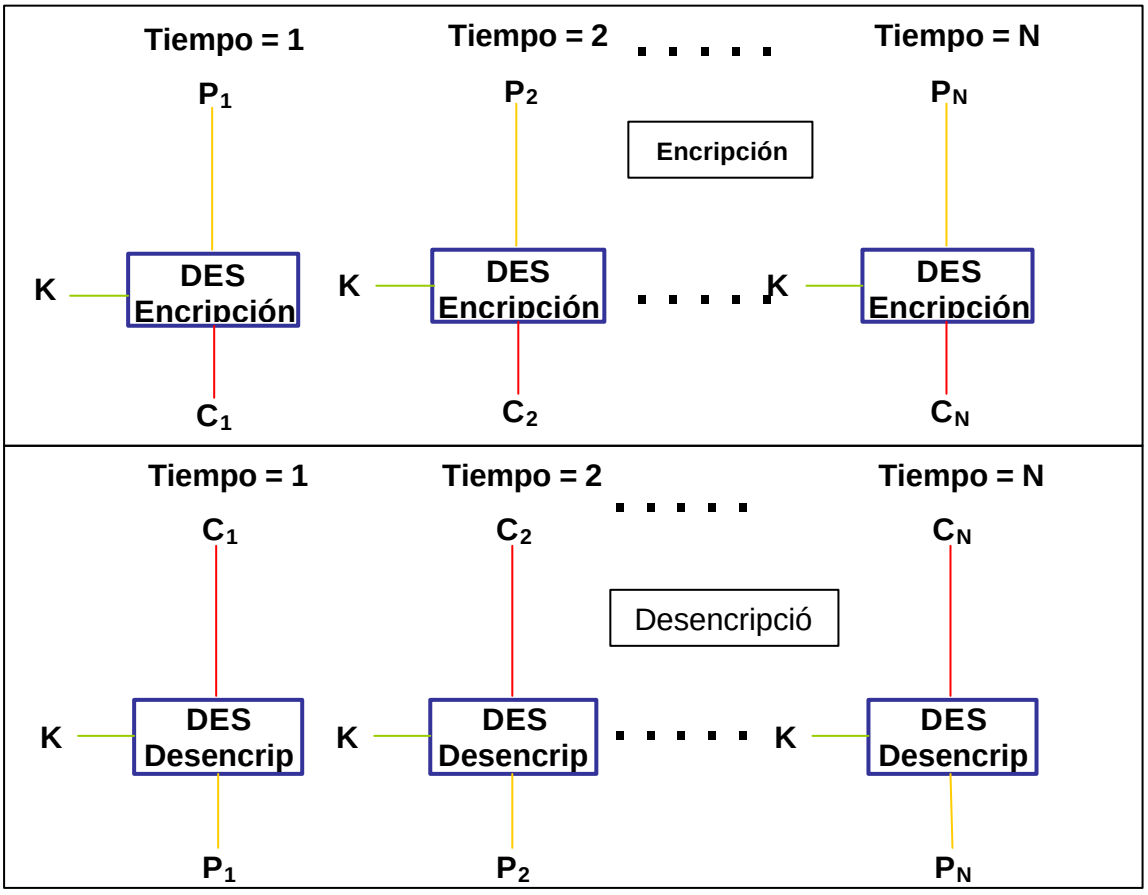


Figura 15 Modo de Operación ECB

Modo cipher block chaining (CBC). **En este esquema, la entrada al algoritmo de encriptación es el resultado del XOR del texto plano actual y el bloque de texto cifrado precedente. La misma clave es usada para cada uno de los bloques. Para la Desencripción, cada bloque cifrado es pasado a través del algoritmo de Desencripción. El resultado sirve como operando a un XOR con el bloque de texto cifrado precedente, para producir el bloque de texto plano. Para ver como funciona, podemos escribir:**

$$C_n = E_K[C_{n-1} \oplus P_n]$$

ENTONCES,

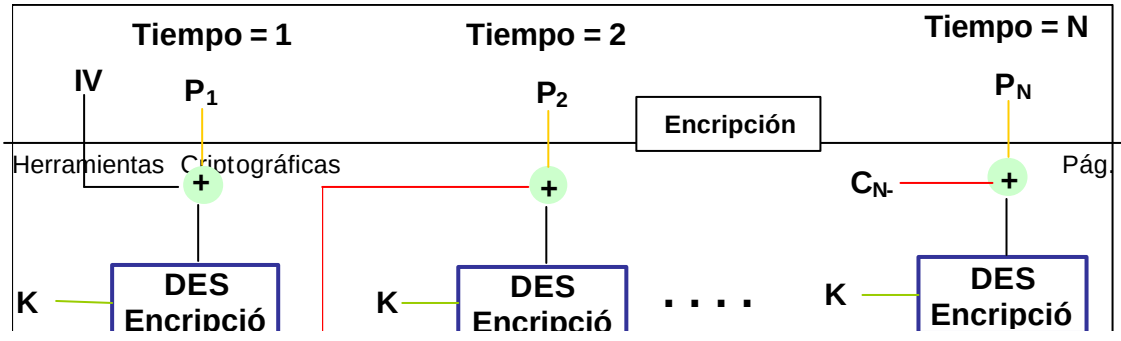
$$D_K[C_n] = D_K[E_K(C_{n-1} \oplus P_n]$$

$$D_K[C_n] = C_{n-1} \oplus P_n$$

$$C_{n-1} \oplus D_K[C_n] = C_{n-1} \oplus C_{n-1} \oplus P_n = P_n$$

Para producir el primer bloque de texto cifrado, un vector de inicialización entra a un XOR con el primer bloque de texto plano. En la desencripción, el vector de inicialización sirve como entrada a un XOR con la salida del algoritmo de desencripción. De esta manera, se recupera el primer bloque de texto plano.

El vector de iniciación debe ser conocido por el emisor y el receptor. Para máxima seguridad, el vector de inicialización debe ser protegido tanto como la clave.



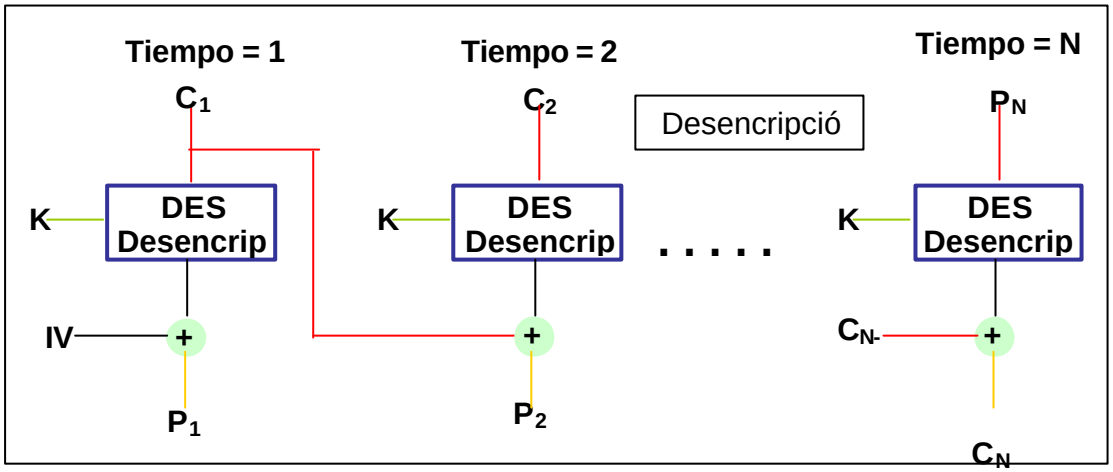


Figura 16 Modo de Operación CBC

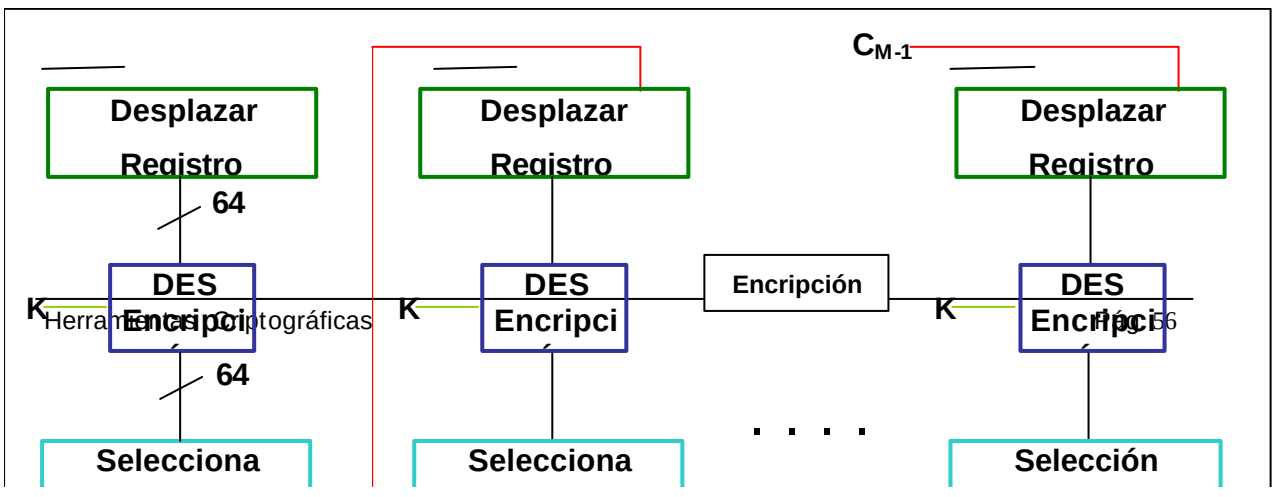
Modo Cipher feedback (CFB). **El DES es un cifrador de bloques, sin embargo, es posible convertir DES en un cifrador de flujo, usando el modo de cipher feedback o el modo output feedback. Un cifrador de flujo elimina la necesidad de rellenar un mensaje para ser un número integral de bloques. Él también puede operar en tiempo real. Esto es, si un flujo de caracteres está siendo transmitido, cada carácter puede ser Encriptado y transmitido inmediatamente, usando un cifrador de flujo orientado a caracteres.**

Una propiedad deseable de un cifrador de flujo, es que el texto cifrado sea de la misma longitud que el texto plano. Esto es, si caracteres de 8 bits están siendo transmitidos, cada carácter debe ser encriptado usando 8 bits. Si más de 8 bits son usados, la capacidad de transmisión es desperdiciada.

La *Figura 17* siguiente muestra el esquema CFB. Se asume que la unidad de transmisión es j bits. Un valor común de j es 8.

Consideremos el proceso de encriptación. La entrada a la función de encriptación es un registro de desplazamiento de 64 bits que es inicialmente establecido a algún vector de iniciación. Los j bits de más a la izquierda (los más significativos) de la salida de la función de encriptación, sirven como entrada a una operación XOR, conjuntamente con la primera unidad de texto plano P_1, para producir la primera unidad de texto cifrado C_1, el cual es entonces transmitido.

Adicionalmente, el contenido del registro de desplazamiento, es desplazado a la izquierda j bits, y C_1 es colocado en la parte más a la derecha (menos significativo) j bits del registro de desplazamiento. Este proceso continúa hasta que todas las unidades del texto plano han sido Encriptado. Para la Desencripción, el mismo esquema es usado, excepto que el texto cifrado recibido entra a un XOR con la salida de la función de encriptación, para producir la unidad de texto plano.



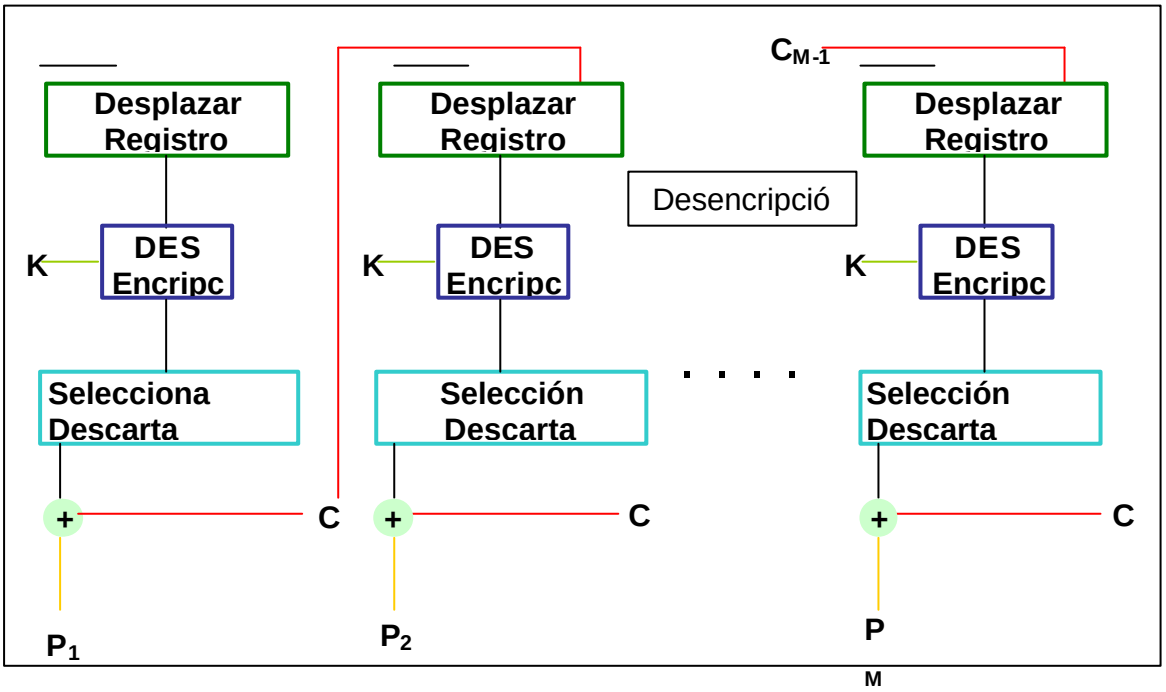
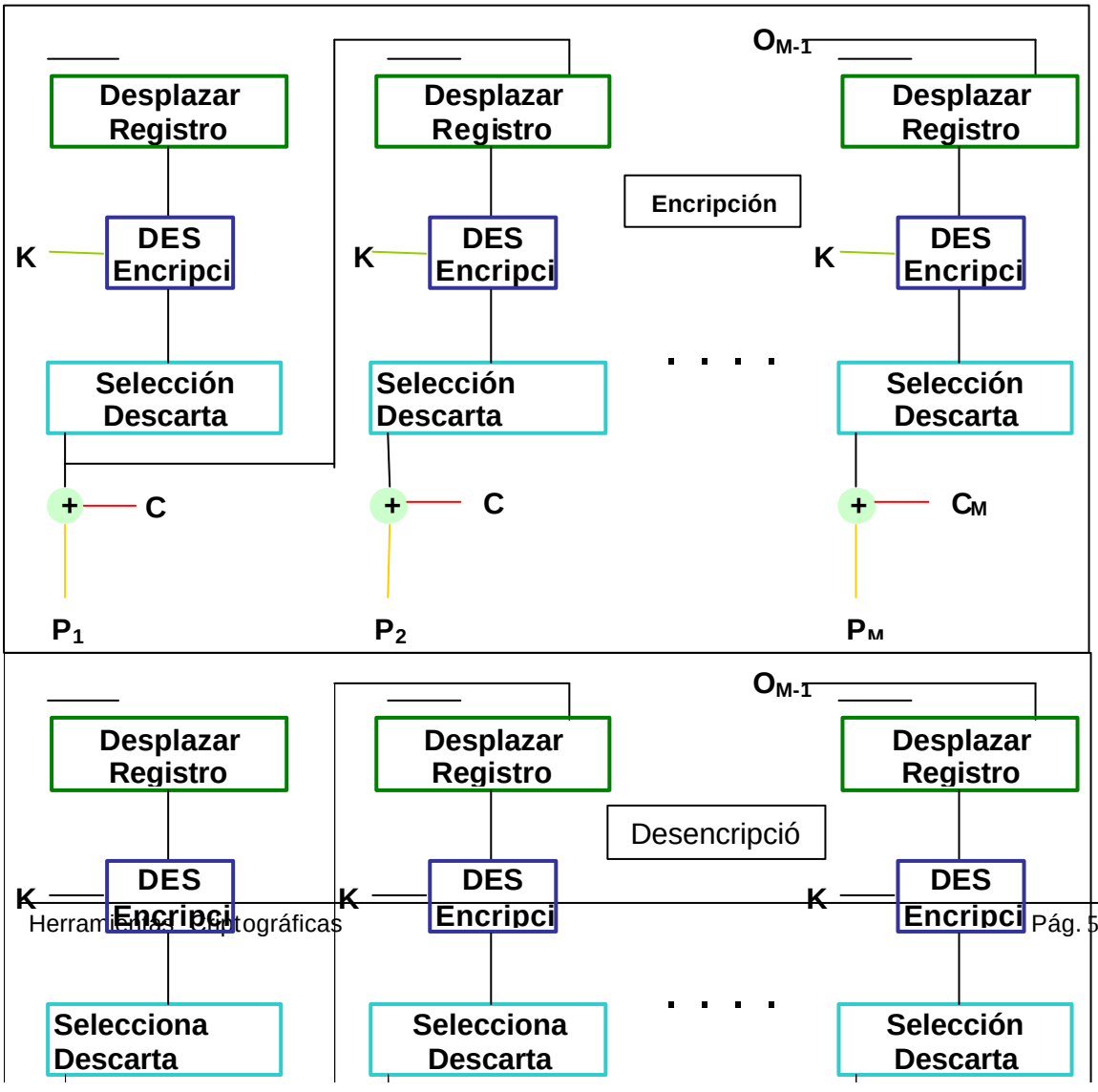


Figura 17 Modo de Operación CFB

Modo output feedback (OFB). **Este modo es similar en su estructura a CFB, como se puede ver en la *Figura 18* Como allí se aprecia, la salida de la función de encriptación es la retroalimentación del registro de desplazamiento en el modo OFB, mientras que en CFB, la unidad de texto cifrado, sirve como retroalimentación al registro de desplazamiento.**

Una ventaja del modo OFB es que los bits de errores en la transmisión, no se propagan. La desventaja de OFB es que parece más vulnerable a ataques por modificación del flujo de mensaje que el modo CFB.



2.3.1.1.4 DOBLE DES

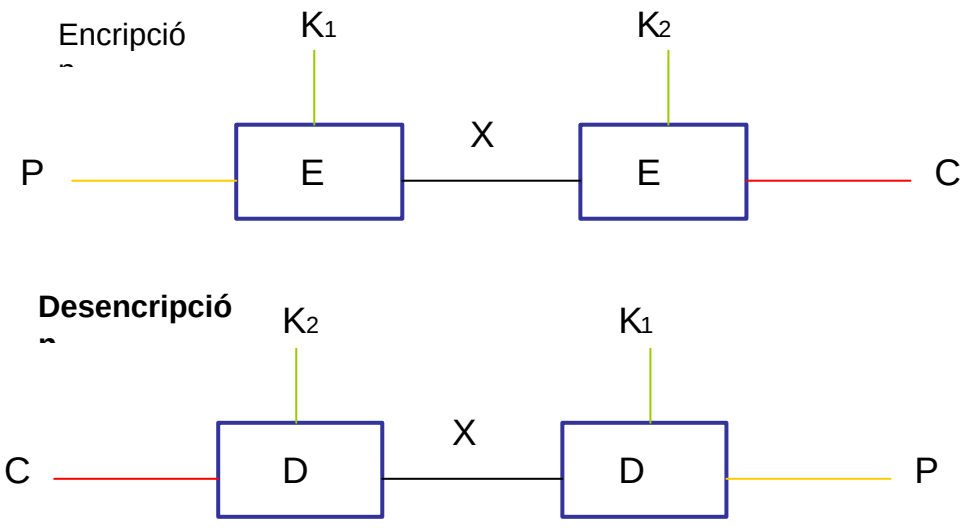


Figura 19 Doble DES

2.3.1.1.5 TRIPLE DES

Para Encriptar con DES Triple, se generan tres claves de 56 bits. Se realiza una Encripción DES de los datos con la primera clave, después se Desencriptan con la segunda, y se vuelven a Encriptar con la tercera. Para Desencriptar se sigue el proceso inverso. Tenemos así un algoritmo de excelente diseño con un total de 2^{112} claves para probar en un ataque de fuerza bruta (sí, deberían ser 2^{168}, pero determinados ataques hacen que el número efectivo de claves sea 2^{112}).

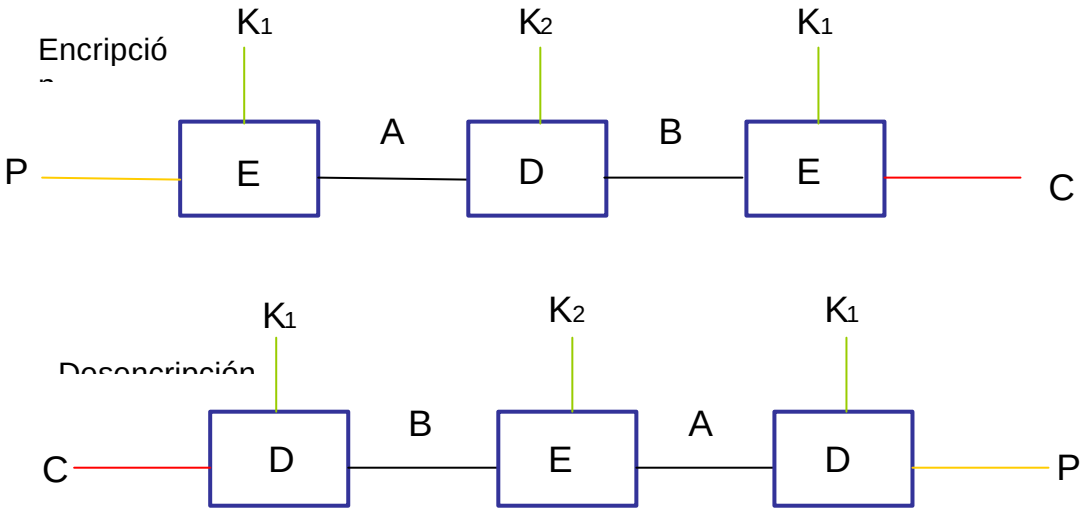


Figura 20 Triple DES

2.3.1.2 CRIPTOANALISIS

Se podría decir que el criptoanálisis se comenzó a estudiar seriamente con la aparición del DES. Mucha gente desconfiaba (y aun desconfía) del

algoritmo propuesto por la NSA. Se dice que existen estructuras extrañas, que muchos consideran sencillamente puertas traseras colocadas por la Agencia para facilitarles la decodificación de los mensajes. Nadie ha podido aun demostrar ni desmentir este punto. Lo único cierto es que el interés por buscar posibles debilidades en el ha llevado a desarrollar técnicas que posteriormente han tenido éxito con otros algoritmos.

Ni que decir tiene que estos métodos no han conseguido doblegar a DES, pero si representan mecanismos significativamente más eficientes que la fuerza bruta para criptoanalizar un mensaje. Los dos métodos que vamos a comentar parten de que disponemos de grandes cantidades de pares texto plano cifrado obtenidos con la clave que queremos descubrir.

2.1.1.2.1 CRIPTOANALISIS DIFERENCIAL

Descubierto por Biham y Shamir en 1990, permite efectuar un ataque de texto plano escogido a DES que resulta mas eficiente que la fuerza bruta. Se basa en el estudio de los pares de criptogramas que surgen cuando se codifican dos textos planos con diferencias particulares, analizando la evolución de dichas diferencias a lo largo de las rondas de DES.

Para llevar a cabo un criptoanálisis diferencial se toman dos mensajes cualesquiera (incluso aleatorios) idénticos salvo en un numero concreto de

bits. Usando las diferencias entre los textos cifrados, se asignan probabilidades a las diferentes claves de cifrado. Conforme tenemos mas y mas pares, una de las claves aparece como la mas probable. Esa será la clave buscada.

2.1.1.2.2 CRIPTOANALISIS LINEAL

El criptoanálisis lineal, descubierto por Mitsuru Matsui, basa su funcionamiento en tomar algunos bits del texto plano y efectuar una operación XOR entre ellos, tomar algunos del texto cifrado y hacerles lo mismo, y finalmente 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 criptogramas diferentes podemos ver si se obtienen mas ceros o mas unos.

Existen combinaciones de bits que, bien escogidas, dan lugar a un sesgo significativo en la medida anteriormente definida, es decir, que el numero de ceros (o unos) es apreciablemente superior. Esta propiedad nos va a permitir poder asignar mayor probabilidad a unas claves sobre otras y de esta forma descubrir la clave que buscamos.

2.3.1.3 IMPLEMENTACION

Se puede realizar de muchas maneras, una de ellas es mediante una función, la cual recibe los siguientes parámetros:

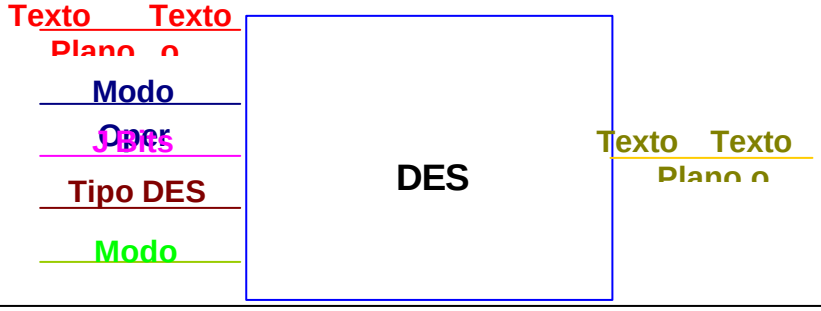


Figura 21 Implementación Cifrador DES

Texto Plano o Texto Cifrado, **dependiendo del *modo* en que se encuentre se introduce ya sea al texto plano o el cifrado, mediante una cadena o un archivo especificando su nombre, la salida de la función también depende del modo, como se puede notar en la gráfica.**

Modo Oper, **indica el modo de operación con el cual se ejecutara el DES. Los valores que puede tomar y el modo de operación que indican estos valores son los siguientes.**

ModoOper=0: Indica que se ejecutara el Modo de Operación ECB
ModoOper=1: Indica que se ejecutara el Modo de Operación CBC
ModoOper=2 : Indica que se ejecutara el Modo de Operación CFB
ModoOper=3 : Indica que se ejecutara el Modo de Operación OFB

J Bits, **Contiene la cantidad de Bits con la cual se deben trabajar los modo de operación CFB y OFB según la teoría de estos modo de operación**

Tipo DES, **indica las variantes DES con la cual se desea trabajar. Sus valores y la explicación son los siguientes:**

TipoDES=0: Ejecuta el modo sencillo del DES.

TipoDES=1: Ejecuta el modo Doble DES.

TipoDES=2: Ejecuta el modo Triple DES.

Modo, **indica si el algoritmo se utiliza ya se para Encriptar (Modo=0) o Desencriptar (Modo<>1).**

*Clave1, Clave2***, estas variables contiene la representación de los 8 caracteres que conforman las claves con las cuales trabajan cada uno de los tipos de DES.**

*Clave3***, Esta variable contiene la representación de los 8 caracteres que conforman la clave con la cual trabajan cada uno de los modos de operación.**

2.3.2 IDEA

El IDEA*(El Algoritmo Internacional De Encripción De Dato)* es un algoritmo de encriptación convencional de dato orientado a bloque, desarrollado por Xuejia Lai y James Massey y presentada su versión original en1990. Una versión revisada, diseñada para ser más fuerte al ataque criptanalítico diferencial fue presentada en 1991.

El algoritmo de clave única(Simétrica) IDEA utiliza texto en bloques de 64 bits y una clave de 128 bits. Ha sido diseñado de tal manera que el proceso de Encripción consiste en ocho pasos de encriptación que son idénticos excepto en los subbloques de la clave utilizados,

terminando con una transformación de la salida. En cada paso se utilizan tres operaciones, suma bit a bit, multiplicación bit a bit y OR exclusivo.

2.3.2.1 DESCRIPCION

Se crean para cada paso de encriptación seis subbloques de 16 bits de la clave de 128 bits, además de otros cuatro para la transformación final, con lo cual tenemos 52 bloques. El proceso para crearlos es el siguiente, la clave de 128 bits es dividida en ocho bloques de 16 bits, que serán los ocho primeros subbloques. La clave es desplazada 25 posiciones a la izquierda, y vuelve a ser dividida en ocho bloques, que serán los siguientes ocho bloques. Este proceso se repite hasta obtener los 52 bloques.

En el primer paso de encriptación el bloque de texto de 64 bits se divide en cuatro bloques de 16 bits, ya que todas las operaciones se harán con números de 16 bits. Se comienza combinando los cuatro primeros subbloques de la clave con dos de los bloques de texto sumándolos bit a bit, y con los otros dos multiplicándolos bit a bit. Por último, se realiza el OR exclusivo entre los bloques resultado de la operaciones anteriores y otros dos subbloques de la clave. Así, al final del primer paso se obtienen cuatro bloques que serán la entrada del segundo paso con el orden cambiado parcialmente.

Este proceso se repite en cada uno de los siete siguientes pasos utilizando diferentes subbloques de la clave. Para terminar, los cuatro bloques resultado del octavo paso son combinados con los cuatro subbloques que quedan sin utilizar sumándolos y multiplicándolos bit a bit, obteniéndose finalmente cuatro bloques de 16 bits de texto Encriptado. En ningún punto del proceso de Encripción se usan contiguamente dos operaciones iguales.

El proceso de Desencripción es prácticamente el mismo que el de encriptación, con la diferencia de los 52 subbloques de la clave son los inversos de los empleados en la encriptación respecto de la operación en la que fueron usados, además de utilizarse en el orden inverso.

Los objetivos de diseño de IDEA pueden ser agrupados en los siguientes.

2.3.2.1.1 FORTALEZAS

Las siguientes características de IDEA están relacionadas con su fortaleza criptográfica:

- ***Longitud del bloque*. El tamaño del bloque debe ser lo suficientemente grande para disuadir el análisis estadístico. Esto es, para denegar a cualquier oponente la ventaja de que algunos bloques aparezcan muchas veces más que otros. Por otra parte, la complejidad de implementación de una función de encriptación efectiva crece exponencialmente con el tamaño del bloque. El uso de un bloque de tamaño 64 bits es reconocido como suficientemente fuerte. Además el uso de un modo de operación CFB fortalece este aspecto del algoritmo.**

- ***Longitud de la clave*. La longitud de la clave debe ser lo suficientemente grande para prevenir efectivamente la búsqueda exhaustiva de la clave. Con una longitud de 128 bits, IDEA parece ser seguro en esta área.**

- ***Confusión*. El texto cifrado debe depender del texto plano y de la clave en una forma complicada y enredada. El objetivo es complicar la determinación de cómo las estadísticas del texto cifrado dependen de las**

estadísticas del texto plano. IDEA alcanza este objetivo usando tres operaciones diferentes.

- ***Difusión*. Cada bit de texto plano debe influenciar cualquier bit de texto cifrado, y cada bit de la clave debe influenciar cualquier bit de texto cifrado. El algoritmo IDEA es muy efectivo en este aspecto.**

En IDEA, la confusión **es alcanzada por la mezcla de tres operaciones diferentes. Cada operación es ejecutada sobre dos operandos de 16 bits, para producir una sola salida de 16 bits. Las operaciones son:**

- ***Un or-exclusivo bit a bit, denotado con $\mathring{A}$.***
- ***Adición de enteros modulo 2^{16} (modulo 65536) con entradas y salidas tratadas como enteros de 16 bits(sin signo), esta operación se denota ⊠.***
- ***Multiplicación de enteros modulo 2^{16} +1 (modulo 65537) con entradas y salidas como enteros de 16 bits(sin signo), excepto un bloque con todos los bits ceros usa la representación de 2^{16}, esta operación es denotada como ⊙.***

Por ejemplo
0000000000000000 ⊙ 1000000000000000 = 1000000000000001
Porque
2^{16} X 2^{15} mod (2^{16}+1)= 2^{15}+1

La tabla muestra valores para las 3 operaciones, operando en números de 2 bits (en vez de 16 bits) estas tres operaciones son incompatibles en el sentido que:

- ***Ningún par de las 3 operaciones satisface la ley distributiva.***

  Ejemplo : $a \boxtimes (b \odot c) \neq (a \boxtimes b) \odot (a \boxtimes c)$

- ***Ningún par de estas operaciones satisface una ley asociativa.***

  Ejemplo : $a \boxtimes (b \boxplus c) \neq (a \boxtimes b) \boxplus c$

Tabla 10 Funciones usadas en IDEA para operadores de 2 bits de longitud

X		Y		X ⊠ Y		X ⊙ Y		X ⊞ Y	
Decimal	**Binario**	**Decimal**	**Binario**	**Decimal**	**Binario**	**Decimal**	**Binario**	**Decimal**	**Binario**
0	00	0	00	0	00	1	01	0	00
0	00	1	01	1	01	0	00	1	01
0	00	2	10	2	10	3	11	2	10
0	00	3	11	3	11	2	10	3	11
1	01	0	00	1	01	0	01	1	01
1	01	1	01	2	10	1	01	0	00
1	01	2	10	3	11	2	10	3	11
1	01	3	11	0	00	3	11	2	10
2	10	0	00	2	10	3	11	2	10
2	10	1	01	3	11	2	10	3	11
2	10	2	10	0	00	0	00	0	00
2	10	3	11	1	01	1	01	1	01
3	11	0	00	3	11	2	10	3	11
3	11	1	01	0	00	3	11	2	10
3	11	2	10	1	01	1	01	1	01
3	11	3	11	2	10	0	00	0	00

El uso combinado de estas tres operaciones, por separado provee una compleja transformación de salida, asiendo el criptoanálisis mucho mas

difícil, a diferencia del DES el cual solo se basa en XOR, en IDEA la difusión es proveída por la construcción básica de bloque del algoritmo conocido como la multiplicación / adicción (MA) el cual se muestra en la siguiente *Figura 22*.

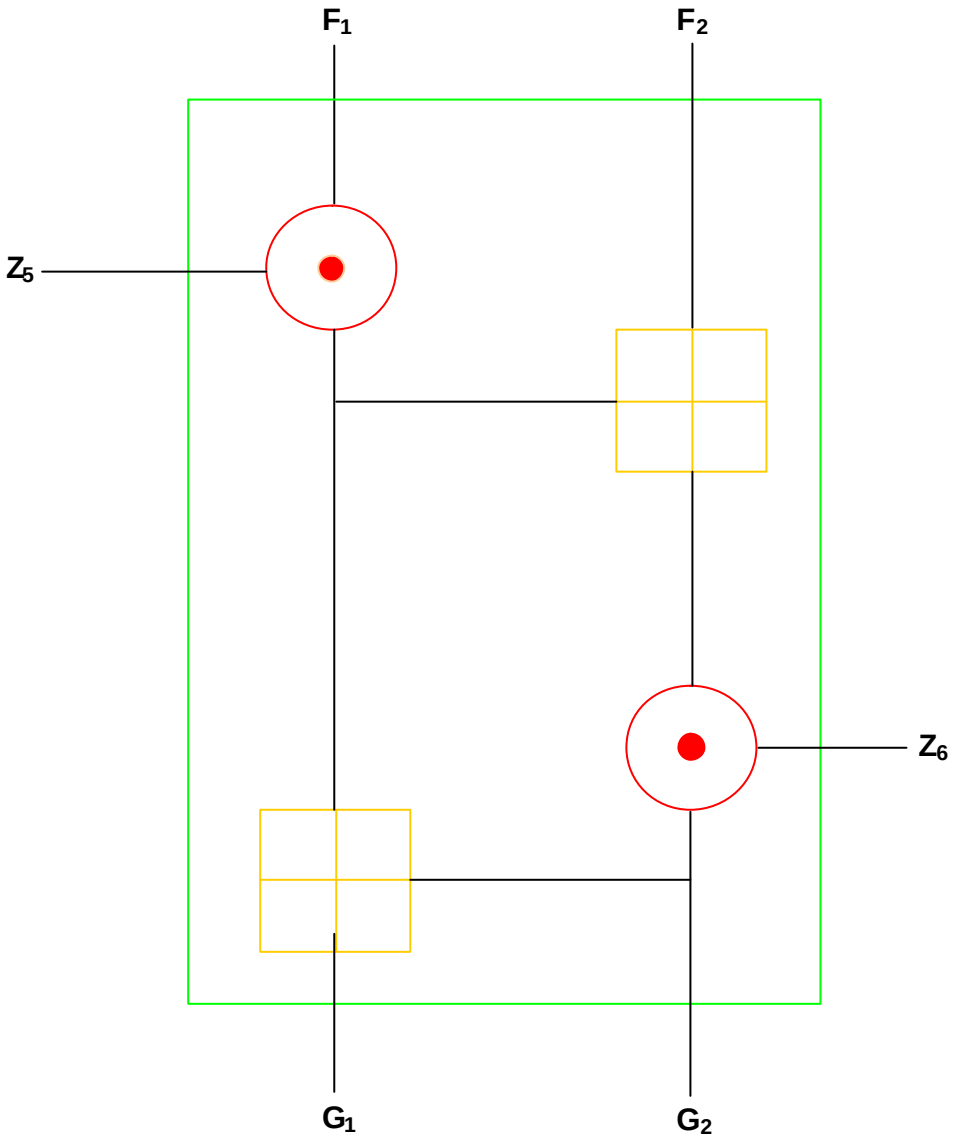


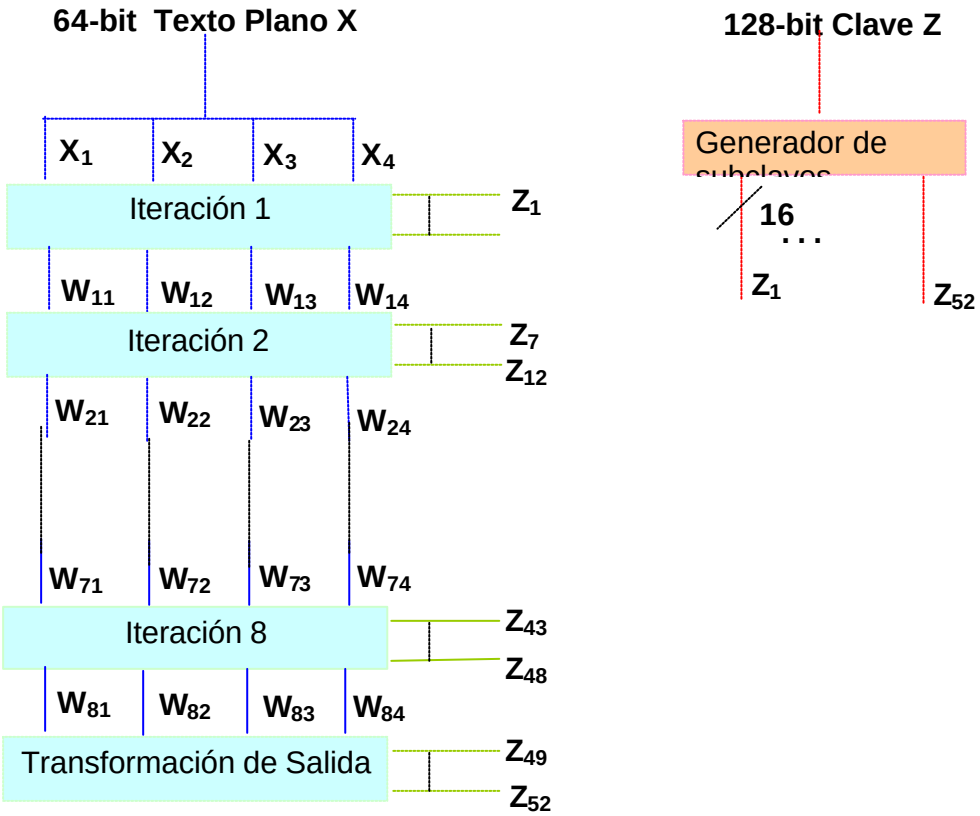
Figura 22 Estructura De Multiplicación / Adición (MA) En IDEA

Esta estructura tiene como entrada 2 valores 16 bits derivados del texto plano y dos subclaves de 16 bits derivadas de la clave, produciendo dos salidas de 16 bits. Un exhaustivo chequeo de un computador a determinado que cada bit de salida de la primera iteración depende en cada bit de entrada derivado del texto plano, y en cada bit de la subclave.

Esta particular estructura es repetida 8 veces en el algoritmo produciendo una muy efectiva difusión, además puede ser demostrado que esta estructura utiliza el menor numero de operaciones(4) requeridas para alcanzar una completa difusión.

2.3.2.1.2 ENCRIPCION

El esquema general para la encriptación en IDEA es ilustrado en la figura 23 siguiente



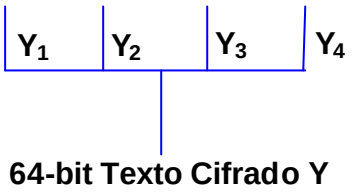


Figura 23 Estructura General Del IDEA

encriptación: el texto plano a ser Encriptado y la clave. El texto plano es de 64 bits de longitud y la clave es de 128 bits de longitud.

El algoritmo de IDEA consiste de 8 iteraciones, seguida de una función de transformación final. El algoritmo parte la entrada en 4 subbloques de 16 bits. En cada una de las iteraciones se toman 4 subbloques de 16 bits cada uno y se produce como salida 4 subbloques de 16 bits cada uno.

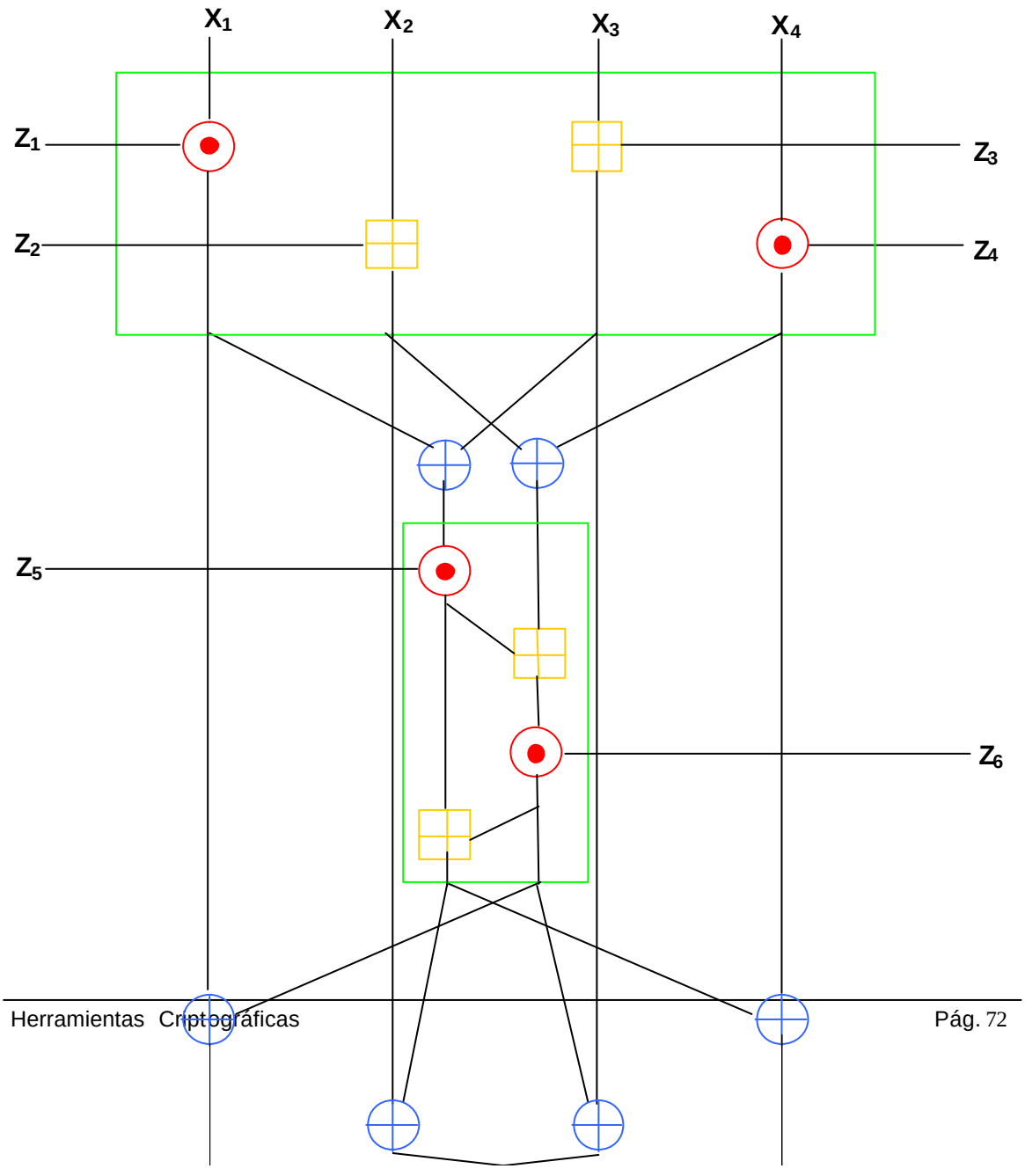
La transformación final también produce 4 subbloques de 16 bits, los cuales son concatenados para formar el texto cifrado de 64 bits. Cada una de las iteraciones hace uso, también, de seis subclaves de 16 bits, mientras que la transformación final usa solo 4 subclaves, para un total de 52 subclaves. Todas estas subclaves son generadas a partir de la clave original de 128 bits.

2.3.2.1.2.1 ITERACIONES

Los detalles de una iteración, son ilustrados en la siguiente *Figura 24.* De hecho, esta figura muestra la primera iteración. Las iteraciones

subsecuentes tienen la misma estructura, pero con subclaves diferentes y el texto plano de entrada derivado.

La iteración comienza con una transformación que combina los cuatro subbloques de entrada con cuatro subclaves, usando las operaciones de adición y multiplicación. Los cuatro bloques de salida de esta transformación son combinados usando la operación XOR para formar dos bloques de 16 bits que son la entrada a la estructura MA (*Ver Figura 2.6*). La estructura MA toma, también, dos subclaves como entrada y combina estas entradas para producir dos salidas de 16 bits.



Finalmente, los cuatro bloques de salida de la transformación inicial son combinados con los dos bloques de salida de la estructura MA, usando la operación XOR, para producir los cuatros bloques de salida para esta iteración.

El noveno estado del algoritmo, rotulado como el estado de transformación de salida es mostrado en la *Figura 25*.

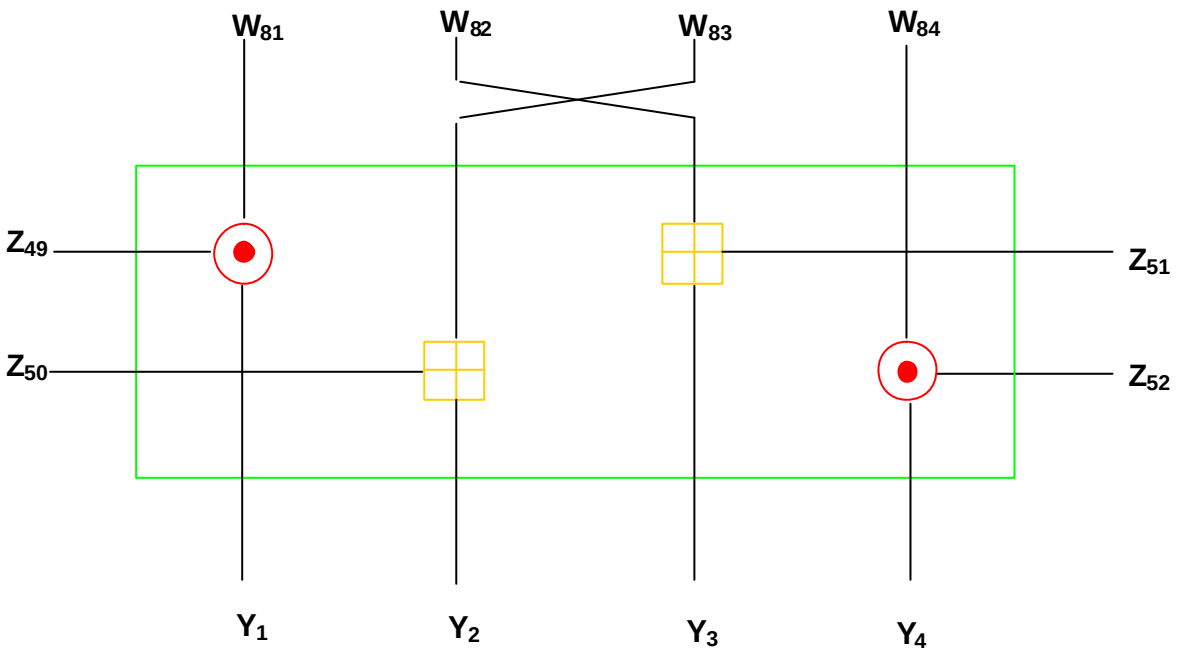


Figura 25 Compuerta De Transformación De Salida De IDEA

Nótese que tiene la misma estructura de la transformación inicial de las iteraciones precedentes. La única diferencia es que la segunda y tercera entrada son intercambiadas, antes de aplicárseles las operaciones. En realidad, esto tiene el efecto de anular el intercambio en el final de la octava iteración.

La razón para este intercambio extra, es que la Desencripción tenga la misma estructura que la Encripción. Otra diferencia, también, es que la novena iteración, solo requiere de cuatro subclaves de entrada, comparada con las seis subclaves de entrada que requieren las ocho primeras iteraciones.

2.3.2.1.2.2 GENERACION DE SUBCLAVES

Observando la *Figura 23*, podemos analizar que 52 subclaves de 16 bits, son generadas a partir de 128 bits de la clave de Encripción. El esquema para la generación es como sigue. Las primeras ocho subclaves, etiquetadas como Z_1, Z_2,...,Z_8, son tomadas directamente de la clave original, con Z_1 igual a los 16 primeros bits (los más significativos), Z_2 corresponde a los 16 bits siguientes y así sucesivamente.

Después de esto, un desplazamiento circular a la izquierda de 25 bits de posiciones es aplicado a la clave y las siguientes ocho subclaves son extraídos. Este procedimiento es repetido hasta que todas las 52 subclaves.

Si la clave como un todo es etiquetada como Z[1..128], entonces las primeras clave de las ocho iteraciones tienen los siguientes bits de asignación:

Z_1 = Z [1.....16]	Z_{25} = Z [76..91]
Z_7 = Z [97.....12]	Z_{31} = Z [44..59]
Z_{13}= Z [90...105]	Z_{37} = Z [37..52]
Z_{19}= Z [83.....98]	Z_{43} = Z [30..45]

2.3.2.1.3 DESENCRIPCION

El proceso de Desencripción es esencialmente el mismo proceso de Encripción. La Desencripción es alcanzada, usando el texto cifrado como entrada a la misma estructura general de IDEA, mostrada en la Figura 23*, pero con una selección diferente de subclaves. Las subclaves de Desencripción U_1, ... ,U_{52} son derivadas de las subclaves de encriptación, como sigue:*

- **Las primeras cuatro subclaves de Desencripción de la iteración i son derivadas de las primeras cuatro subclaves de encriptación de la iteración (10 – i), tomando el estado de transformación, como la iteración nueve. La primera y cuarta subclaves de Desencripción son iguales al inverso multiplicativo módulo (2^{16}+1) de las correspondientes primera y segunda subclaves de encriptación. Para las iteraciones 2 hasta la 8, las segundas y terceras subclaves de Desencripción, son iguales al inverso aditivo módulo (2^{16}) de las correspondientes terceras y segundas subclaves de encriptación. Para las iteraciones 1 y 9, las segundas y terceras subclaves de Desencripción, son iguales al inverso aditivo módulo (2^{16}) de las correspondientes segundas y terceras subclaves de encriptación.**

- ***Para las primeras ocho iteraciones, las últimas dos subclaves de Desencripción de la iteración i, son iguales a las últimas dos subclaves de encriptación de la iteración (9-i).***

La *Tabla 11* resume estas relaciones. Para el inverso multiplicativo, la notación usada es Z_j^{-1}, para el inverso aditivo, la notación usada es –Z.

Tabla 11 Subclaves De Encripción y Desencripción				
	Encriptación		Desencripción	
Estado	Designación	Equivalente a	Designación	Equivalente a
Iteración1	$Z_1Z_2Z_3Z_4Z_5Z_6$	Z[1..96]	$U_1U_2U_3U_4U_5U_6$	$Z_{49}^{-1} -Z_{50} -Z_{51}Z_{52}^{-1}Z_{47}Z_{48}$
Iteración2	$Z_7Z_8Z_9Z_{10}Z_{11}Z_{12}$	**Z[97..128;26..89]**	$U_7U_8U_9U_{10}U_{11}U_{12}$	$Z_{43}^{-1} -Z_{45} -Z_{44}Z_{46}^{-1}Z_{41}Z_{42}$
Iteración3	$Z_{13}Z_{14}Z_{15}Z_{16}Z_{17}Z_{18}$	**Z[90..128;1..25;51..82]**	$U_{13}U_{14}U_{15}U_{16}U_{17}U_{18}$	$Z_{37}^{-1} -Z_{39} -Z_{38}Z_{40}^{-1}Z_{35}Z_{36}$
Iteración 4	$Z_{19}Z_{20}Z_{21}Z_{22}Z_{23}Z_{24}$	**Z[83..128;1..50]**	$U_{19}U_{20}U_{21}U_{22}U_{23}U_{24}$	$Z_{31}^{-1} -Z_{33} -Z_{32}Z_{34}^{-1}Z_{29}Z_{30}$
Iteración 5	$Z_{25}Z_{26}Z_{27}Z_{28}Z_{29}Z_{30}$	**Z[76..128;1..43]**	$U_{25}U_{26}U_{27}U_{28}U_{29}U_{30}$	$Z_{25}^{-1} -Z_{27} -Z_{26}Z_{28}^{-1}Z_{23}Z_{24}$
Iteración 6	$Z_{31}Z_{32}Z_{33}Z_{34}Z_{35}Z_{36}$	**Z[44..75;101..128;1..36]**	$U_{31}U_{32}U_{33}U_{34}U_{35}U_{36}$	$Z_{19}^{-1} -Z_{21} -Z_{20}Z_{22}^{-1}Z_{17}Z_{18}$
Iteración 7	$Z_{37}Z_{38}Z_{39}Z_{40}Z_{41}Z_{42}$	**Z[37..100,126..128;1..29]**	$U_{37}U_{38}U_{39}U_{40}U_{41}U_{42}$	$Z_{13}^{-1} -Z_{15} -Z_{14}Z_{16}^{-1}Z_{11}Z_{12}$
Iteración 8	$Z_{43}Z_{44}Z_{45}Z_{46}Z_{47}Z_{48}$	**Z[30..125]**	$U_{43}U_{44}U_{45}U_{46}U_{47}U_{48}$	$Z_7^{-1} -Z_2 -Z_8Z_{10}^{-1}Z_5Z_6$
Transformación	$Z_{49}Z_{50}Z_{51}Z_{52}$	**Z[23..86]**	$U_{49}U_{50}U_{51}U_{52}$	$Z_1^{-1} -Z_2 -Z_3Z_4$

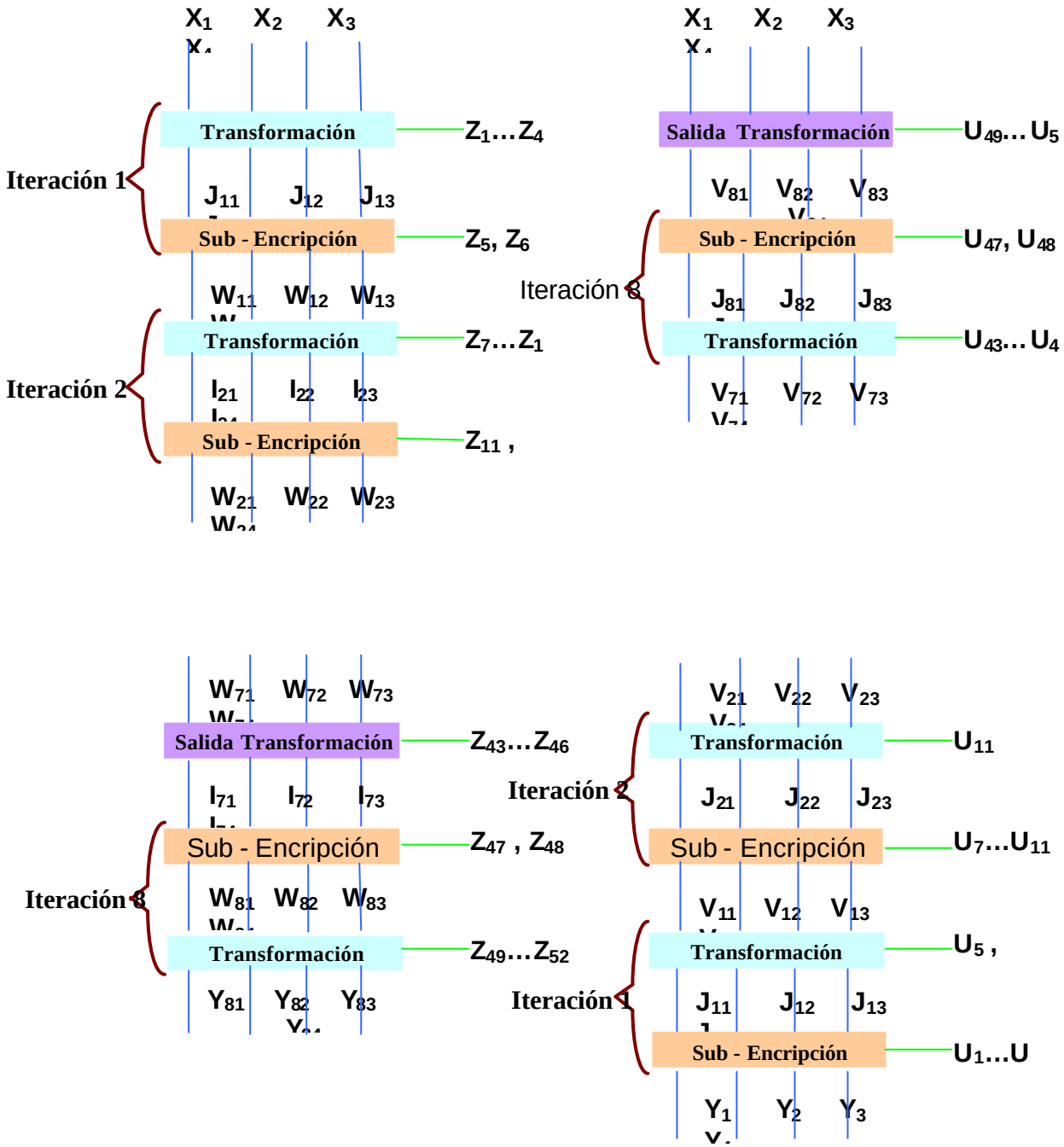


Figura 26 Encripción y Desencripción Del IDEA

2.3.2.1.4 MODOS DE OPERACIÓN

Al igual que DES, cuatro modos de operación pueden ser usados con IDEA:

Modo Electronic codebook (ECB). **Cada bloque de 64 bits de texto plano es codificado independientemente usando la misma clave. Esta técnica es útil para encriptar pequeños bloques de datos. Él podría ser usado para encriptar claves IDEA de 128 bits**

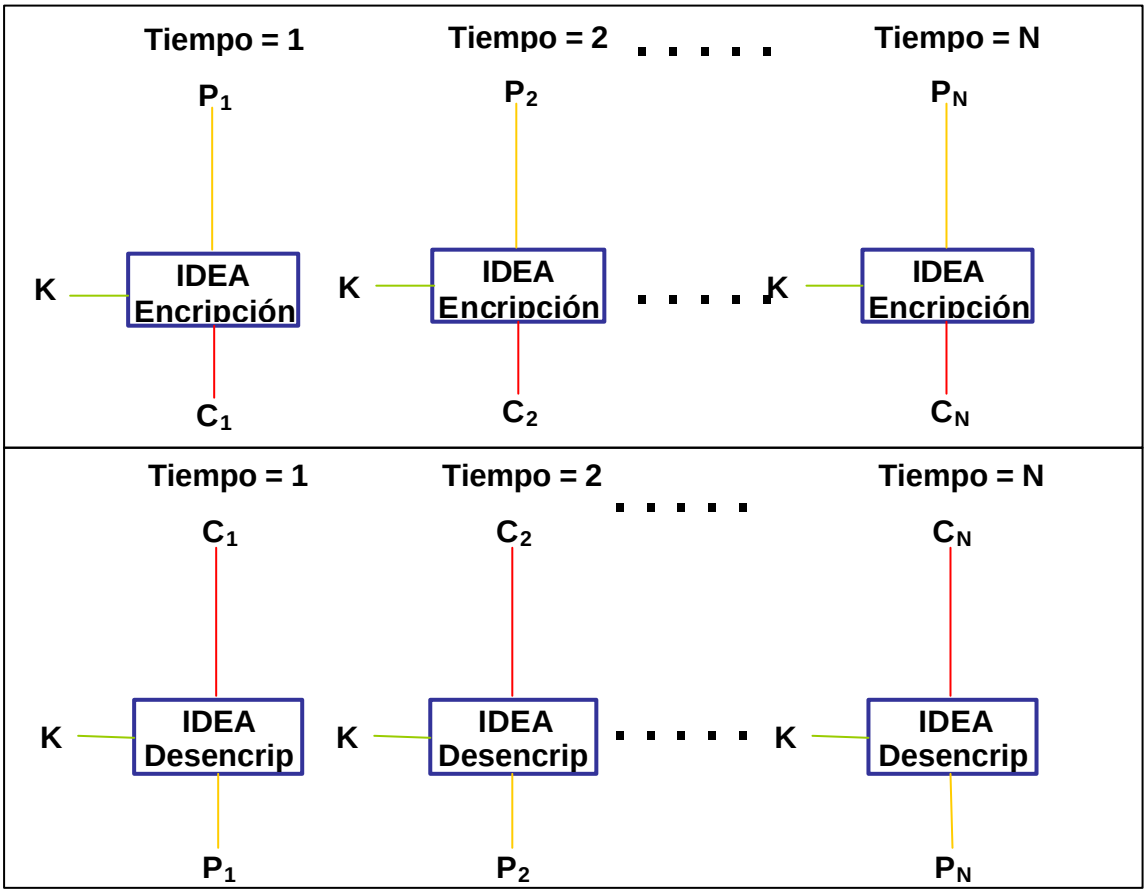


Figura 27 Modo de Operación ECB para él IDEA

Modo Cipher block chaining (CBC). **La entrada al algoritmo de encriptación es el XOR de los siguientes 64 bits de texto plano y los 64 bits precedentes de texto cifrado.**

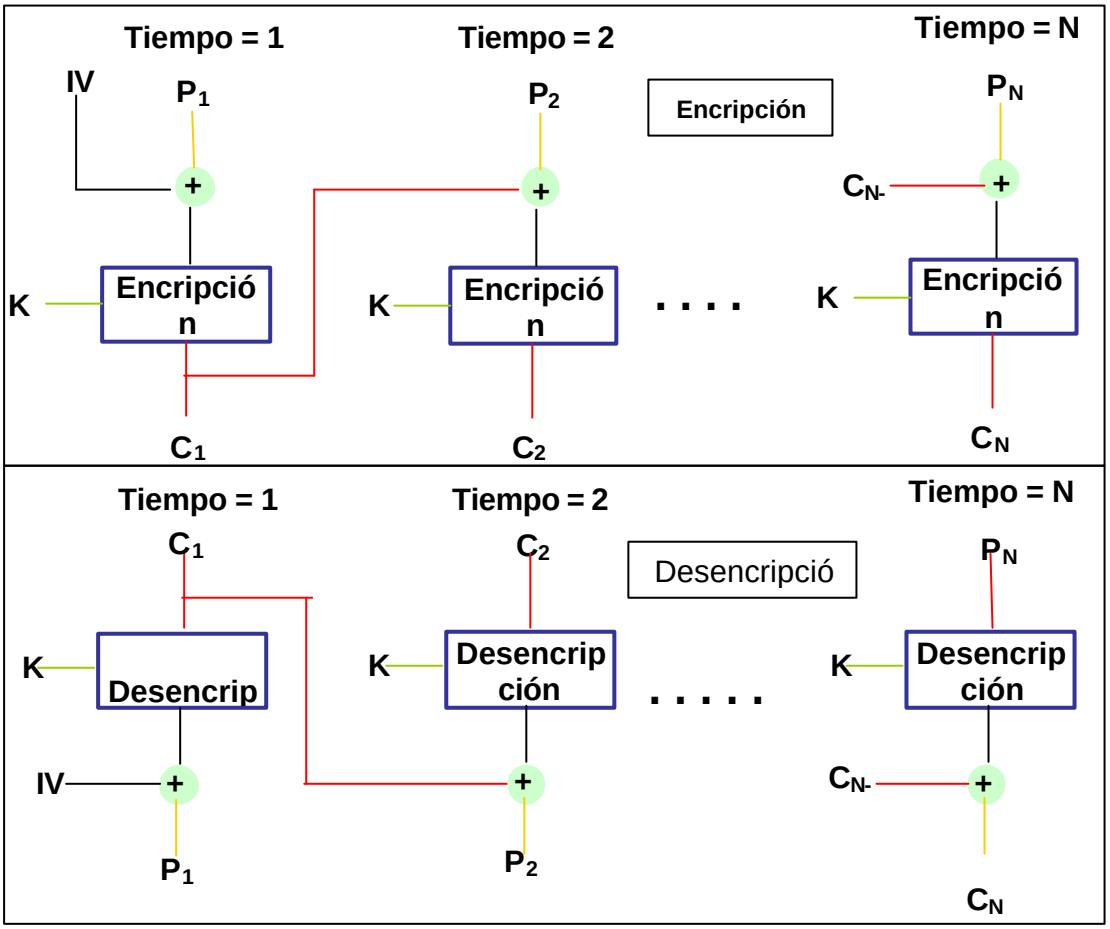


Figura 28 Modo de Operación CBC para el IDEA

Modo Cipher feedback (CFB). **La entrada es procesada j bits en un momento. El texto cifrado precedente, es usado como entrada al algoritmo de encriptación para producir salida seudoaleatoria, la cual sirve como entrada a un XOR con el texto plano para producir la siguiente unidad de texto cifrado. Este modo es útil para codificar bloques grandes de entrada**

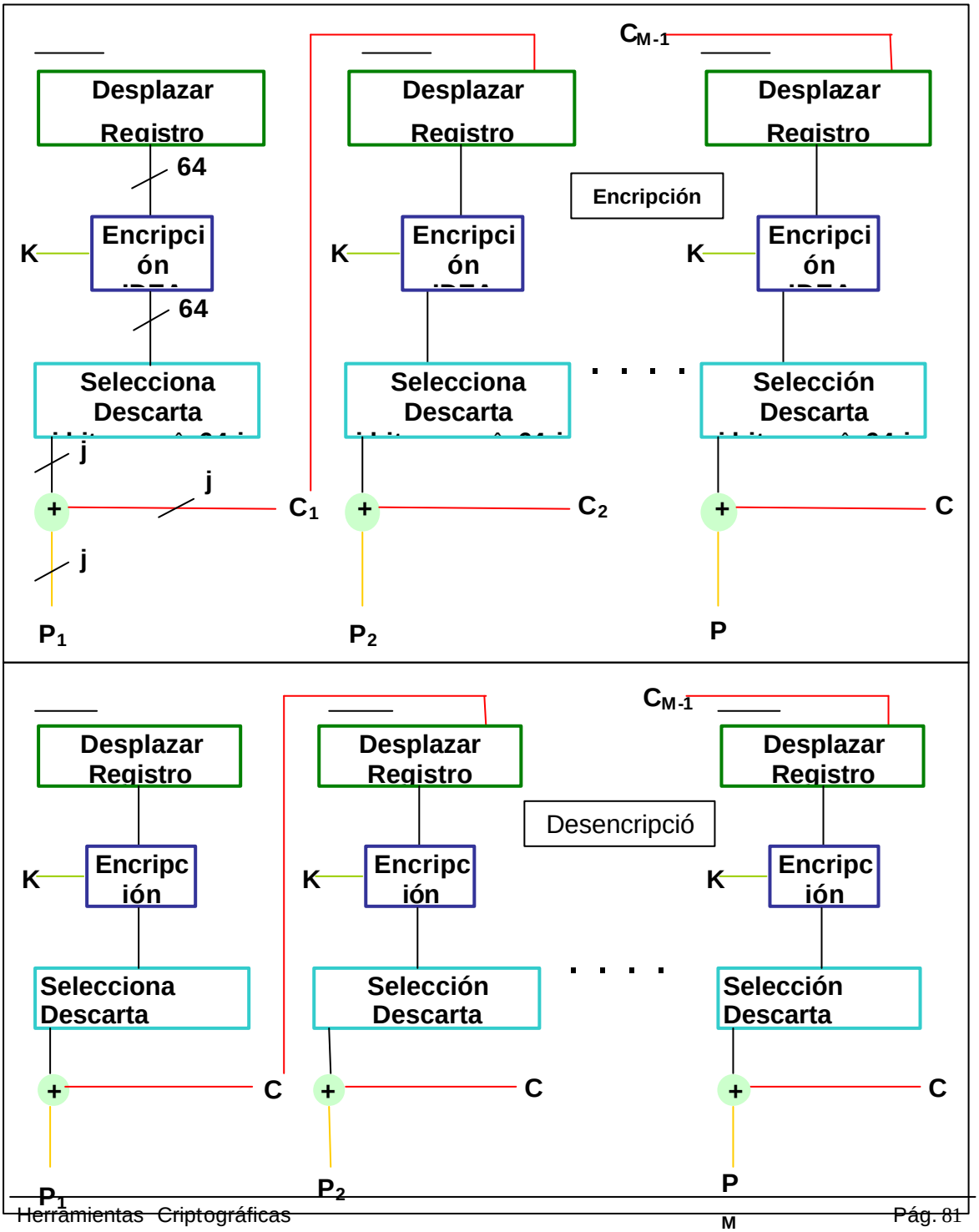


Figura 29 Modo de Operación CFB Para él IDEA

Modo Output feedback (OFB). **Similar a CFB, excepto que la entrada al algoritmo de encriptación es la salida precedente de IDEA. Este modo es útil para transmisiones orientadas a flujo sobre un canal ruidoso.**

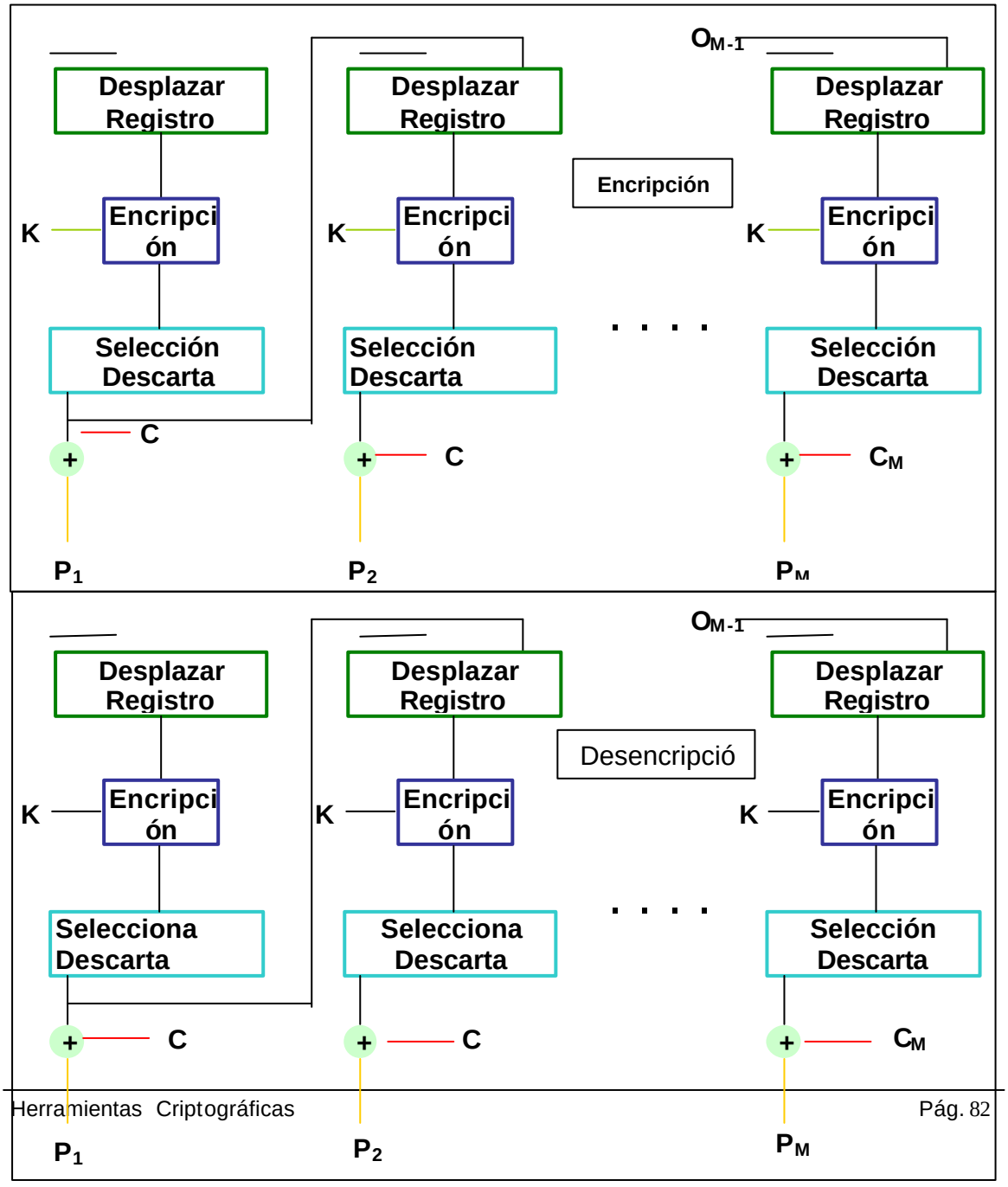


Figura 30 Modo de Operación OFB Para IDEA

2.3.2.2 CRIPTOANALISIS

El largo de la clave del IDEA es 128 bits, dos veces tan largo como el DES. Asumiendo que un ataque de fuerza bruta es lo más eficiente, requeriría 2128 encripciones para recuperar la clave. Diseñar un chip que pueda probar un billón de claves por segundo y arrojar un billón de ellas al problema, y tomara todavía 1013 años que es mas largo que la edad del universo. Un arreglo de 1024 chips de este tipo puede encontrar la clave en un día, pero no hay suficiente átomos de silicón en el universo para construir una maquina de este tipo. Tal vez la fuerza bruta no es la mejor manera de atacar el IDEA.

El algoritmo es demasiado nuevo para dar un resultado de criptoanálisis definitivo. Los diseñadores han hecho su mejor trabajo para hacer su algoritmo inmune al criptoanálisis diferencial; ellos definieron el concepto de cifrador de Markov y mostraron que la resistencia al criptoanálisis diferencial puede ser ordenada y cuantificada

Willy Meller examino las tres operaciones algebraicas del IDEA, y apuntó que mientras que sean incompatibles hay casos donde ellas pueden ser simplificados en una manera para facilitar el criptoanálisis en un porcentaje de tiempo. Su ataque es más eficiente que el de la fuerza bruta por dos ciclos.

Joan Daemen descubrió una clase de llaves débiles para el IDEA. Estas no son claves débiles en el caso de DES. Son débiles en el caso que si son usadas, un atacante puede fácilmente identificarlas escogiendo un ataque al texto plano. Por ejemplo:
000,0000,0x00,0000,00000,000x,xxxx,x000

Él numero en la posición de x puede ser cualquier numero. En cualquier caso, la posibilidad de generar una de estas claves débiles es muy pequeña: 1 en 296. No hay peligro si usted escoge estas claves aleatoriamente. Y es fácil modificar IDEA para que no tenga ninguna llave débil.

2.3.2.3 IMPLEMENTACION

Se puede realizar de muchas maneras, una de ellas es mediante una función, la cual recibe los siguientes parámetros:

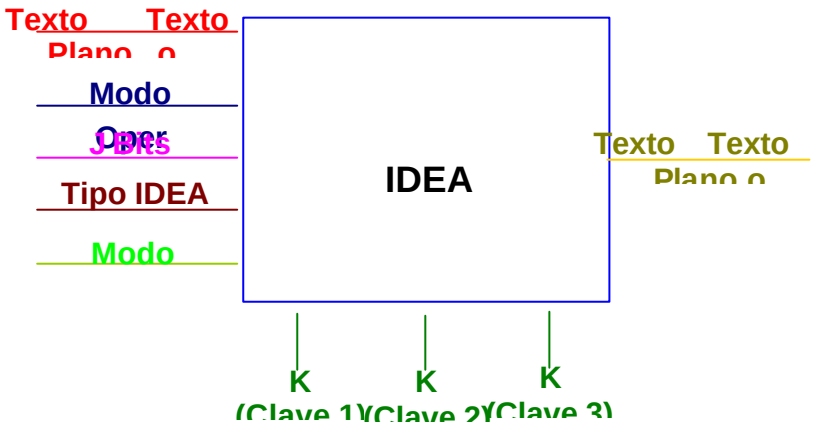


Figura 31 Implementación Cifrador IDEA

Texto Plano o Texto Cifrado, **dependiendo del *modo* en que se encuentre se introduce ya sea al texto plano o el cifrado, mediante una cadena o un archivo especificando su nombre, la salida de la función también depende del modo, como se puede notar en la gráfica.**

Modo Oper, **indica el modo de operación con el cual se ejecutara el IDEA. Los valores que puede tomar y el modo de operación que indican estos valores son los siguientes.**

ModoOper=0: Indica que se ejecutara el Modo de Operación ECB
ModoOper=1: Indica que se ejecutara el Modo de Operación CBC
ModoOper=2 : Indica que se ejecutara el Modo de Operación CFB
ModoOper=3 : Indica que se ejecutara el Modo de Operación OFB

J Bits, **Contiene la cantidad de Bits con la cual se deben trabajar los modo de operación CFB y OFB según la teoría de estos modo de operación**

Tipo IDEA, **indica las variantes IDEA con la cual se desea trabajar. Sus valores y la explicación son los siguientes:**

TipoIDEA=0: Ejecuta el modo sencillo del IDEA.
TipoIDEA=1: Ejecuta el modo Doble IDEA.
TipoIDEA=2: Ejecuta el modo Triple IDEA.

Modo, **indica si el algoritmo se utiliza ya se para Encriptar (Modo=0) o Desencriptar (Modo<>1).**

*Clave1, Clave2***, estas variables contiene la representación de los 8 caracteres que conforman las claves con las cuales trabajan cada uno de los tipos de IDEA.**

*Clave3***, Esta variable contiene la representación de los 8 caracteres que conforman la clave con la cual trabajan cada uno de los modos de operación.**

3. ENCRIPCION DE CLAVE PUBLICA

3.1 ALGORITMOS DE CRIPTOGRAFIA ASIMETRICA

Los algoritmos de clave pública, o algoritmos asimétricos, han demostrado su interés para ser empleados en redes de comunicación inseguras (Internet). Introducidos por Whitfeld Diffe y Martin Hellman a mediados de los años 70, su novedad fundamental con respecto a la criptografía simétrica es que las claves no son únicas, sino que forman pares. Hasta la fecha han aparecido multitud de algoritmos asimétricos, la mayoría de los cuales son inseguros.

Otros son poco prácticos, bien sea porque el criptograma es considerablemente mayor que el mensaje original, bien sea porque la longitud de la clave es enorme. Se basan en general en plantear al atacante problemas matemáticos difíciles de resolver. En la practica muy pocos algoritmos son realmente útiles. Él más popular por su sencillez es RSA, que ha sobrevivido a multitud de ataques, si bien necesita una longitud de clave considerable.

Los algoritmos asimétricos emplean generalmente longitudes de clave mucho mayores que los simétricos. Por ejemplo, mientras

que para algoritmos simétricos se considera segura una clave de 128 bits, para algoritmos asimétricos se recomiendan claves de al menos 1024 bits. Además, la complejidad de calculo que comportan estos últimos los hace considerablemente más lentos que los algoritmos de cifrado por bloques. En la practica los métodos asimétricos se emplean únicamente para codificar la clave de sesión (simétrica) de cada mensaje.

Los algoritmos asimétricos poseen dos claves diferentes en lugar de una, *Kp* y *KP*, denominadas clave privada y clave publica. Una de ellas se emplea para codificar, mientras que la otra se usa para decodificar. Dependiendo de la aplicación que le demos al algoritmo, la clave publica será la de cifrado o viceversa. Para que estos criptosistemas sean seguros también ha de cumplirse que a partir de una de las claves resulte extremadamente difícil calcular la otra.

Antes todos los Sistemas Criptográficos se basaban en permutaciones y sustituciones y ahora se utilizan funciones matemáticas. La seguridad de los Criptosistemas de asimétricos depende de la longitud de la clave, lo que se deriva exponencialmente en trabajo computacional. La más importante

característica de los Criptosistemas de Clave Pública es que es computacionalmente no factible determinar la clave de Desencripción cuando se conoce únicamente el algoritmo de Encripción y la clave de Encripción.

A continuación se verá el modelo general de un Criptosistema de Clave Pública, se hará una comparación con los algoritmos de Encripción convencionales, las aplicaciones y requerimientos de este modelo así como su huso mas frecuente..

3.1.1 MODELO

En la siguiente *Figura 32* se ilustra el proceso de encripción de clave pública, en donde cuatro pasos esencialmente se deben cumplir.

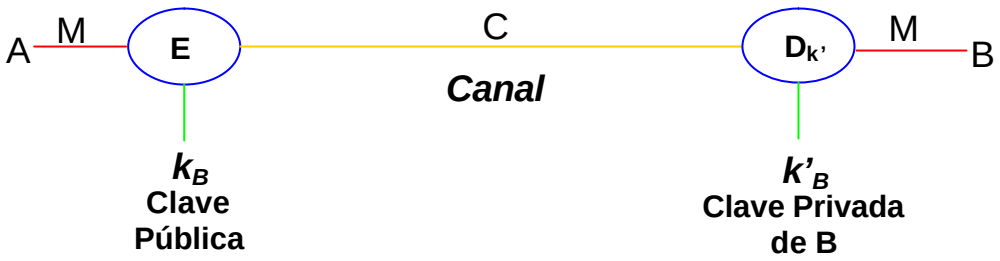


Figura 32 Modelo de Cifradores de Clave Publica

Los cuatro pasos son:

I. **Cada punto final en un Sistema genera un par de claves para ser usadas en la encripción de mensajes y en la desencripción de criptogramas recibidos.**

II. **Cada sistema publica su clave de Encripción en un registro público o un archivo. Esta será la clave pública y la otra la clave privada.**

III. **Si A desea enviar un mensaje a B, este Encripta el mensaje usando la clave pública de B.**

IV. Cuando B recibe el mensaje, este lo Desencripta usando su clave privada. Nadie más puede Desencriptar el mensaje porque solamente B conoce su clave privada

3.1.2 COMPARACION CON LOS CRIPTOSISTEMAS TRADICIONALES

El siguiente cuadro nos resume esta comparación:

Tabla 12 ***Cifradores Convencionales Vs Cifradores de Clave Publica***	
Criptosistemas Convencionales Simétricos	Criptosistemas de Clave Pública **Asimétricos**
***Necesidades para trabajar*:** • **El mismo algoritmo es usado con la misma clave para la Encripción y la Desencripción.** • **El transmisor y el receptor deben compartir el algoritmo y la clave.**	***Necesidades para trabajar:*** • **Un algoritmo es usado para Encripción y Desencripción con un par de claves, una para la Encripción y otra para la Desencripción.** • **El transmisor y receptor deben cada uno tener una del par de claves complementarias.**
Necesidades para seguridad: • **La clave debe ser secreta.** • **El mismo algoritmo es usado con la misma clave para la Encripción y la Desencripción.** • **El conocimiento del algoritmo más los criptogramas son insuficiente para determinar la clave.**	***Necesidades para seguridad:*** • **Una de las dos claves debe ser secreta.** • **Es casi imposible Desencriptar un mensaje si ninguna información adicional se tiene disponible.** • **El conocimiento del algoritmo más una de las claves más los criptogramas deben ser insuficiente para determinar la otra**

	clave.

3.1.3 APLICACIONES

Los Criptosistemas de clave pública tienen 3 aplicaciones importantes:

- ***Encripción /Desencripción :*** **El transmisor Encripta el mensaje con un recipiente de claves públicas.**

- ***Firma digital:*** **El transmisor envía un mensaje con su clave privada. La firma es generalmente un pequeño bloque de datos del mensaje.**

- ***Intercambio de claves:*** **Dos lados cooperan para intercambiar una sesión de claves. Muchos enfoques diferentes son posibles, involucrando las claves privadas de una de ambas partes.**

3.2 LUC

3.2.1 DESCRIPCION

Es un Sistema de Clave Pública desarrollado por un grupo de investigadores. Como RSA puede ser utilizado para Encripción, firma e intercambio de claves. Es similar al RSA, pero debido a que fue desarrollado fuera de los Estados Unidos no está sujeto a controles de exportación fuera de dicho país. LUC está basado en enteros grandes en una secuencia LUCAS

3.2.1.1 FILOSOFIA

Las secuencias de LUCAS son dos series de enteros Un y Vn generados por dos enteros P y Q. La teoría general de estas series fue primeramente desarrollada por Eduardo Lucas en 1878. El enfoque principal de interés en las secuencias de Lucas ha sido probado para probar números primos.

3.2.1.1.1 DEFINICION DE SECUENCIA DE LUCAS

Seleccionar dos enteros no negativos, P y Q, y considerar la siguiente ecuación cuadrática

$$X^2 - PX + Q = 0$$

Las raíces de esta ecuación son $(P \pm (P^2 - 4Q)^{1/2})/2$; de donde la expresión $(P^2 - 4Q)$ es conocida como el discriminante D. Ahora se etiquetan las raíces como a y ß como sigue

$$a = (P + \sqrt{D}) / 2 \qquad y \qquad ß = (P - \sqrt{D}) / 2$$

Por lo que se puede tener las siguientes relaciones entre a y ß

$$a + ß = P$$

$$(a)(ß) = Q$$

$$a - ß = \sqrt{D}$$

Estas relaciones ya están probadas en la matemática común y no es necesario hacerlo aquí.

Ahora, se asume que se tiene un valor de *P* y *Q* tal que *D* ¹ *0*, para definir la secuencia de Lucas como sigue:

$$U_n(P,Q) = (\alpha^h - \beta^h) / (\alpha - \beta)$$

$$V_n(P,Q) = \alpha^h + \beta^h \qquad \text{Para } n >= 0 \text{ y } (\alpha - \beta) \neq 0$$

Es fácil ver que:

$$U_o(P,Q) = 0$$
$$V_o(P,Q) = 2$$
$$U_1(P,Q) = 1$$
$$V_1(P,Q) = P$$

Se puede reescribir las secuencias para mostrar que cada miembro depende solamente de dos miembros precedentes. Considere:

$$P(\alpha^{n-1} + \beta^{n-1}) - Q(\alpha^{n-2} + \beta^{n-2}) = \alpha^{n-2}(P\alpha - Q) + \beta^{n-2}(P\beta - Q)$$
$$= \alpha^{n-2}[(\alpha+\beta)\alpha - \alpha\beta] + \beta^{n-2}[(\alpha+\beta)\beta - \alpha\beta]$$
$$= \alpha^{n-2}(\alpha^2) + \beta^{n-2}(\beta^2)$$
$$= \alpha h + \beta h$$

De lo anterior se obtienen:

$$V_n(P,Q) = PV_{n-1}(P,Q) - Q\,V_{n-2}(P,Q)$$

Similarmente, se puede mostrar que:

$$U_n(P,Q) = PU_{n-1}(P,Q) - Q\,U_{n-2}(P,Q)$$

La *Tabla 13* muestra las secuencias de *Un* y *Vn* para los valores de P= 3 y Q =1.

Tabla 13 Secuencia de Lucas para P=3 y Q=1		
N	Un(3,1)	Vn(3,1)
0	0	2
1	1	3
2	3	7
3	8	18
4	21	47
5	55	123
6	144	322
7	377	843
8	987	2.207
9	2.584	5.778
10	6.765	15.127
11	17.711	39.603
12	46.368	103.682
13	121.393	271.443
14	317.811	710.647
15	832.040	1.860.498
16	2.178.309	4.870.847
17	5.702.887	12.752.043
18	14.930.352	33.385.282
19	39.088.169	87.403.803
20	102.334.155	228.826.127
21	267.914.296	599.074.578
22	701.408.733	1.568.397.607
23	1.836.311.903	4.106.118.243
24	4.807.526.976	10.749.957.122

3.2.1.1.2 PROPIEDADES DE $V_n(P,Q)$

Considere el desarrollo de la secuencia de la Tabla 13 *modulo de algún entero* N, N>2. *Es claro que los números de LUCAS crecen muy rápidamente. Podría responder la siguiente pregunta: ¿Si se calcula la secuencia de LUCAS para cualquier punto* n *y entonces se toma el resultado modulo* N, *se conseguirá el mismo resultado si se le aplica la operación modulo a cada paso en la secuencia?*

Es decir si se cumple lo siguiente

$$V_n(PmodN,QmodN) = V_n(P,Q)modN$$

La respuesta es SÍ. Y se prueba a continuación: Considere unos ejemplos

$$V_o(PmodN, QmodN) = 2$$
$$(V_o(P,Q))modN = 2\ modN = 2$$
$$V_1(PmodN, QmodN) = PmodN$$
$$(V_1(P,Q))\ modN = PmodN.$$

Se puede aplicar el resultado por inducción a *n*. Entonces; Por definición:

$$V_n(P,Q)modN = [PV_{n-1}(P,Q) - QV_{n-2}(P,Q)]modN.$$

Y por las reglas de la aritmética modular.

$$V_n(P,Q)modN = (PmodN)(V_{n-1}(P,Q)modN) - (QmodN)(V_{n-2}(P,Q)modN)$$

y por inducción

$$V_n(P,Q)modN = (PmodN)V_{n-1}(PmodN, QmodN) - (QmodN)V_{n-2}(PmodN, QmodN)$$

por lo que por la definición de la secuencia de Lucas, sé podrá reescribir el lado derecho así:

$$V_n(P,Q)modN = V_n(PmodN, QmodN)$$

Lo cual era lo que se quería demostrar.

Por otro lado, ahora considere los valores $V_k(P,Q)$ para P y Q^k, para Q, con K entero y positivo. Los valores considerados son enteros y no negativos, por

lo que las elecciones son legales. La ecuación cuadrática quedaría como la siguiente:

$$X^2 + V_k(P,Q)X + Q^k = 0$$

donde, las raíces de esta ecuación serán a' y ß' las cuales satisfacen lo siguiente:

$$a' = (V_k(P,Q) + \{ [V_k(P,Q)]^2 - 4\,Q^k \}^{1/2})/2$$
$$ß' = (V_k(P,Q) - \{ [V_k(P,Q)]^2 - 4\,Q^k \}^{1/2})/2$$

por la definición de las secuencias de LUCAS:

$$V_n(V_k(P,Q), Q^k) = (a')^n + (ß')^n$$

Por otra parte si *n* = *k*, entonces $V_k(P,Q) = a^k + ß^k$

$$(a')(ß') = Q^k = (aß)^k = a^k\, ß^k \quad (\text{porque } aß=Q)$$

Por lo que a' y a^k satisfacen la misma ecuación, lo mismo que $ß^k$ y *ß'*. Por lo tanto:

$a^k = a'$ ***y*** $ß' = ß^k$ **ó viceversa**.

Esto significa que se puede escribir lo siguiente:

$$\begin{aligned} V_n(V_k(P,Q), Q^k) &= (a')^n + (ß')^n \\ &= (a^k)^n + (ß^k)^n \\ &= a^{kn} + ß^{kn} \\ &= V_{nk}(P,Q) \end{aligned}$$

Si se hace a *Q=1*, lo cual es válido ya que 1 es un entero no negativo.

Se Tiene que:

$$V_{nk}(P,1) = V_n[V_k(P,1), 1]$$

3.2.1.2 APLICACIÓN DE LUC A LA CRIPTOGRAFIA DE CLAVE PUBLICA

Suponga que podría encontrar dos enteros, e *y* d, *tal que lo siguiente es verdad*

$$V_{de}(X,1) \equiv XmodN$$

Entonces, por definición

$$V_{de}(XmodN, 1) = V_{de}(X,1)modN = XmodN$$

por lo que

$$V_{de}(XmodN,1) = V_d(V_e(XmodN,1), 1) = XmodN$$

De hecho, se podría tomar un texto plano en forma de un entero *X* menor que *N*, Encriptarlo como $V_e(XmodN,1)$ **y posteriormente Desencriptarlo como** *XmodN*.

Pero antes de decir cuales son esos enteros (*d* y *e*) se definirán otras dos cantidades importantes:

***Símbolo Legendre*.** **Es una cantidad que se define como:**

(D/P) = 0 si P divide a D

= -1 si hay un número x tal que D º x^2 mod P.

= 1 si tal número, x, no existe.

Con cualquier secuencia de Lucas definida por dos números primos diferentes *P* y *Q*, existe una generalización de la función de Euler llamada función de Lehmer.

Para el entero $N = p\,q$, **con *p* y *q* primos diferentes, la función de Lehmer de *N* es:**

$$T(N) = [p-(D/P)]\ [q-(D/q)]$$

Para nuestro propósito no nos interesa el producto pero si el mínimo común múltiplo de los factores, que se expresa así:

$$S(N) = lcm[(p-(D/p);\ q-(D/q)]$$

Ahora, si *N = p q* es un producto de dos números primos diferentes y se selecciona a e **como cualquier número primo relativo a *S(N)*, se puede determinar d como:**

ed = k * S(N) + 1, para algún entero k

lo que es equivalente a:

$$ed \ ^{o}\ 1 \bmod S(N)$$

por otro lado, con varias definiciones se puede llegar a demostrar que

$$X_e(V_d(X, 1),1) = V_d(V_e(X,1),1) = X$$

para un número dado $X<N$.

3.2.1.3 ALGORITMO DE CLAVE PUBLICA DE LUC

La estructura completa de LUC es similar al RSA. El texto plano es encriptado en bloques con cada bloque se tiene un numero binario menor que algún número N. *La Encripción y desencripción son de la forma siguiente para algún texto plano* P *y un texto cifrado* C *(por bloques):*

$$C = V_e(P,1) \bmod N$$
$$M = V_d(C,1) \bmod N$$

Ambos, tanto el emisor como el receptor, deben conocer el valor de *N*, el emisor conoce el valor *e* y únicamente el receptor conoce el valor de *d*. Por lo que se define la clave pública y la clave privada así:

$$KU = \{e,N\} \text{ (pública)}$$
$$KR = \{d,N\} \text{ (privada)}$$

Para que este algoritmo sea satisfactorio se deben cumplir los siguientes requisitos:

- **Debe ser posible encontrar valores de *e*, *d*, N tal que:**

$$V_d(V_e(P,1) \bmod N,1) \bmod N = P \bmod N$$

- **Debe ser relativamente fácil calcular** $X_e(P,1) mod N$ **y** $V_d(C,1) mod N$ **para todos los valores de** ***P<N*** **.**

- **Debe ser imposible determinar d dado *e* y *N*, e imposible determinar *P* dado *C*, *e* y *N*.**

3.2.1.4 LUC EN 9 PASOS

El sistema LUC se realiza en los siguientes siete pasos, en el cual cada usuario calcula sus claves privadas y públicas:

I. **Seleccionar al azar dos números primos grandes (Mayores de 100 dígitos)** ***p*** **y** ***q.***

II. **Calcular** ***n = p.q***

III. **Seleccionar un número entero *e(Clave Publica)* menor que *n* y que cumpla con**
gcd[(p –1)(q-1)(p+1)(q+1), e]=1

IV. **Calcular D*(Discriminante),* así** $D = P^2 - 4$

V. **Calcular S(N), así** $S(N) = \left[\left(p - \left(\frac{D}{p}\right)\right)\left(q - \left(\frac{D}{q}\right)\right)\right]$

VI. **Calcular d*(clave Privada),* así** $d = e^{-1} \bmod S(N)$

VII. **Publicar *KU = (e, N)* como clave pública y guardar KR*(d, N)* como clave secreta.**

VIII. **La transformación asociada a la clave pública es *C = Ve(P,1)(mod N)***

P< N

IX. La transformación asociada a la clave privada es *P = Vd(C, 1)(mod N)*

***Nota* : La selección de números primos grandes se hace generando un número impar aleatorio con el orden de magnitud deseada y después se prueba su primaridad con algoritmos probabilísticos.**

3.2.2 CRIPTOANALISIS

La fuerza criptográfica del LUC depende en dos consideraciones, "*Es Imposible determinar d dado e, N", "Es Imposible determinar P dado C, e, N".*

Ambas de estas preguntas llevan a respuestas que son esencialmente las mimas para el RSA. Para determinar *d* dadas *e* y *N* podría requerirse determinar los dos factores primos de *N.* Esto parece ser incomputacinalmete imposible para grandes valores de N. Además, en el caso de LUC hay cuatro diferentes valores para cada e, N. Donde únicamente una de los cuales trabajara para un P dado por esto la versión LUC del problema parece ser mas difícil que la versión del RSA.

El segundo Problema puede ser atacado por fuerza bruta tratando todos los posibles valores de d. Como hemos visto, la carga computacional para el LUC es similar a la del RSA. Por eso para un tamaño dado de clave, LUC es resistente a los métodos de fuerza bruta como lo es el RSA.

3.2.3 IMPLEMENTACION

Se puede realizar de muchas maneras, una de ellas es mediante una función, la cual recibe los siguientes parámetros:

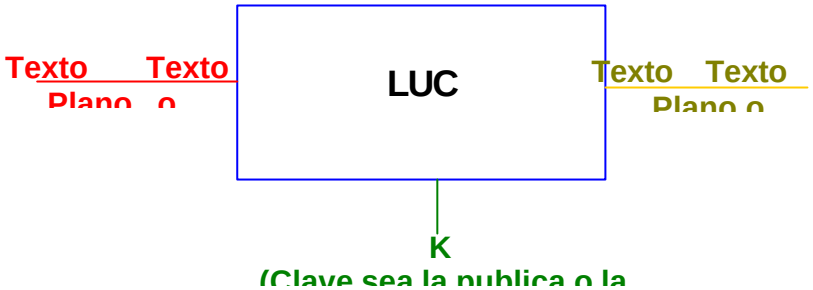


Figura 33 Implementación cifrador LUC

Texto Plano o Texto Cifrado, **dependiendo de lo que se desee la función recibe ya sea le texto plano o el texto cifrado, de igual forma devuelve lo contrario de lo que recibe como se puede notar en la gráfica.**

K o Clave, **la cual puede se vale este algoritmo para poder Encriptar o Desencriptar el archivo de entrada. Su valor máximo depende de tamaño del alfabeto**

3.3 RSA

De entre todos los algoritmos asimétricos, quizás RSA sea el mas sencillo de comprender e implementar. Sus pares de claves son duales, por lo que sirve tanto para privacidad como para autenticar. Su nombre proviene de sus tres inventores: Ron Rivest, Adi Shamir y Leonard Adleman. Desde su nacimiento nadie ha conseguido probar o rebatir su seguridad, pero se le tiene como uno de los algoritmos asimétricos más seguros.
RSA se basa en la dificultad para factorizar grandes números. Las claves publica y privada se calculan a partir de un numero que se obtiene como producto de dos primos grandes. El atacante se enfrentara, si quiere recuperar un texto plano a partir del criptograma y la llave publica, a un problema de factorización

3.3.1 DESCRIPCIÓN

Para generar un par de llaves (*KP*; *Kp*), en primer lugar se escogen aleatoriamente dos números primos grandes, *p* y *q*. Después se calcula el

producto $n = pq$. Escogeremos ahora un numero e primo relativo con $(p - 1)(q - 1)$. $(e; n)$ será la clave publica. Nótese que e debe tener inversa modulo $(p \_ 1)(q \_ 1)$, por lo que existiría un numero d tal que:

$$de \ ^{o} \ 1 \ (mod \ (p - 1)(q - 1))$$

Es decir, que d es la inversa de e modulo $(p - 1)(q - 1)$. $(d; n)$ será la clave privada. Esta

inversa puede calcularse fácilmente empleando el Algoritmo Extendido de Euclides. Nótese que si desconocemos los factores de n, este calculo resulta prácticamente imposible.

La encripción se lleva a cabo según la expresión:

$$c = m^{e} \ (mod \ n)$$

mientras que la desencripción se hará de la siguiente forma:

$$m = c^{d} \ (mod \ n)$$

ya que

$$c^{d} = (m^{e})^{d} = m^{ed} = m^{k(p \_ 1)(q \_ 1)+1} = (m^{k})^{(p \_ 1)(q \_ 1)} m$$

recordemos que $F(n) = (p - 1)(q - 1)$, además, $(m^{k})^{(p - 1)(q - 1)} = 1$, lo cual nos lleva de nuevo a m.

En la practica, cogeremos p y q con un numero grande de bits, por ejemplo 200, con lo que n tendrá 400 bits. Subdividiremos el mensaje que queramos enviar en bloques de 399 bits (de esta forma garantizamos que el valor de cada bloque sea menor que n) y efectuamos la encripción de cada uno. Obtendremos un mensaje cifrado ligeramente más grande, puesto que estará compuesto por bloques de 400 bits. Para desencriptar partiremos el mensaje cifrado en bloques de 400 bits (ya que en este caso sabemos que el valor de cada bloque ha de ser menor que n), y obtendremos bloques de 399 bits. El atacante, si quiere recuperar la clave privada a partir de la publica, debe conocer los factores p y q de n, y esto representa un problema

computacionalmente intratable, siempre que *p* y *q* y, por lo tanto, *n* sean lo suficientemente grandes.

3.3.2 RSA EN SIETE PASOS

El sistema RSA se realiza en los siguientes siete pasos, en el cual cada usuario calcula sus claves privadas y públicas:

I. **Seleccionar al azar dos números primos grandes (Mayores de 100 dígitos) *p* y *q.***

II. **Calcular *n = p.q***

III. **Seleccionar un número entero *e* menor que *n* y primo relativo con $\phi(n)=(p-1)(q-1)$**

IV. **Calcular *d,* tal que $de \equiv 1 \bmod \phi(n)$**

V. **Publicar *P(e, n)* como clave pública y guardar *S(d, n)* como clave secreta.**

VI. **La transformación asociada a la clave pública es $C = P(M, e, n) \equiv M^e \bmod n$**

VII. **La transformación asociada a la clave privada es $M = S(C, d, n) \equiv C^d \bmod n$**

***Nota* : La selección de números primos grandes se hace generando un número impar aleatorio con el orden de magnitud deseada y después se prueba su primaridad con algoritmos probabilísticos.**

3.3.2.1 EJEMPLO

Hagamos un ejemplo para un par de números primos pequeños:

I. ***p=3*** **y** ***q=17***

II. ***n=(3)(17)=51***

III. **Se selecciona** ***e=5*** **para** $f(n)=(2)(16)=32$**, ya que cumple las condiciones** ***e>n*** **y** ***gcd(e,*** f ***(n))=1***

IV. **De la congruencia** $5d \equiv 1 \bmod 32$ **se calcula** ***d=13***

V. **Se publica** ***P(5, 51)*** **como clave pública y se guarda** ***S(13, 51)*** **como clave secreta.**

VI. **El criptograma para un mensaje** ***M*** **(Por ejemplo** ***M=2*****) es** $C \equiv 2^5 \bmod 51=32$

VII. **El mensaje desencriptado es** $M \equiv 32^{13} \bmod 51=2$

3.3.3 FORTALEZAS DEL RSA

Técnicamente no es cierto que el algoritmo RSA deposite su fuerza en el problema de la factorización. En realidad el hecho de tener que factorizar un numero para descifrar un mensaje sin la clave privada es una mera conjetura. Nadie ha demostrado que no pueda surgir un método en el futuro que permita descifrar un mensaje sin usar la clave privada y sin factorizar él modulo *n*. De todas formas, este método podrá ser empleado como una nueva técnica para factorizar números enteros, por lo que la anterior afirmación se considera en la practica cierta. De hecho, existen estudios que demuestran que incluso recuperar solo algunos bits del mensaje original resulta tan difícil como descifrar el mensaje entero.

Aparte de factorizar *n*, se podría intentar calcular F *(n)* directamente, o probar por la fuerza bruta tratando de encontrar la clave privada. Ambos ataques son mas costosos computacionalmente que la propia factorización de *n*, afortunadamente.

Otro punto que cabria preguntarse es que pasaría si los primos *p* y *q* que escogemos realmente fueran compuestos. Recuerde que los algoritmos de prueba de primos que se conocen son probabilísticos, por lo que jamas se tendrá la absoluta seguridad de que *p* y *q* son realmente primos. Por otra parte, si *p* o *q* fueran compuestos, el algoritmo RSA simplemente no funcionara.

3.3.4 CRIPTOANALISIS

Cuatro posibles enfoques de Criptoanálisis pueden ser identificados:

- **Ataque de fuerza bruta: Trata todas las posibles claves privadas**
- **Factorización de *p* en dos factores primos.**
- **Determinar $f(n)$ directamente sin determinar primero *p* ni *q*. Después se determina *d*.**
- **Determinar *d* directamente sin determinar primero $f(n)$.**

Para minimizar riesgos en los posibles ataques deben tener en cuenta las siguientes consideraciones:

- ***n* debe tener de *100* a *200* dígitos**
- ***p* y *q* deben diferir por pocos dígitos en cuanto a su longitud y deben ser del orden de 10^{75} a 10^{150}**
- **Ambos *p-1* y *q-1* deben tener un factor primo grande *gcd(p-1, q-1)* debe ser pequeño.**

3.3.5 IMPLEMENTACION

Se puede realizar de muchas maneras, una de ellas es mediante una función, la cual recibe los siguientes parámetros:

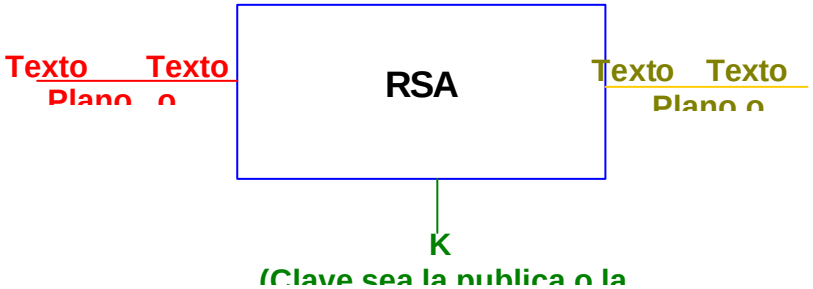


Figura 34 Implementación cifrador RSA

Texto Plano o Texto Cifrado, **dependiendo de lo que se desee la función recibe ya sea le texto plano o el texto cifrado, de igual forma devuelve lo contrario de lo que recibe como se puede notar en la gráfica.**

K o Clave, **la cual puede se vale este algoritmo para poder Encriptar o Desencriptar el archivo de entrada. Su valor máximo depende de tamaño del alfabeto**

4. FUNCIONES HASH

4.1 MD5

Message Digest 5, el algoritmo Resumen de Mensaje MD5 fue desarrollado por Ron Rivest en el MIT.

4.1.1 DESCRIPCION

4.1.1.1 LOGICA DEL MD5

El algoritmo toma como entrada un mensaje de longitud arbitraria y produce como salida un resumen del mensaje de 128 bits. La entrada es procesada en bloques de 512-bits. Este algoritmo esta conformado por los siguientes pasos.

PASO 1. Añadir relleno de bits. ***El mensaje es rellenado para que la longitud en bits sea congruente con 448 módulo 512 (longitud*** ^{o} ***448 mod 512). Esto quiere decir, que la longitud del mensaje relleno es 64 bits menor que un entero múltiplo de 512 bits. El relleno es añadido siempre, aunque el mensaje ya sea de la longitud deseada. Por ejemplo, si el mensaje es de 448 bits de longitud, éste es rellenado con 512 bits, sumando un total de 960 bits. Por lo tanto, el número de bits de relleno se sitúa entre 1 y 512 bits. El relleno consiste en un único bit 1 seguido del número necesario de bits 0.***

PASO 2. Añadir longitud. Una representación de la longitud del mensaje original en 64 bits, sin contar el relleno, es añadida al resultado del paso 1. Si la longitud original es mayor o igual que 2^{64}, entonces sólo serán utilizados lo s 64 bits de la parte baja de longitud original. De esta manera, el campo contiene la longitud original del mensaje, módulo 2^{64}. El resultado de los dos primeros pasos produce un mensaje de longitud múltiplo de 512 bits(numero entero).

En la *Figura 35*, el mensaje extendido es representado como una secuencia de bloques de 512 bits $Y_0, Y_1, \ldots, Y_{L-1}$, por lo que la longitud total del mensaje extendido es L·512 bits. Lo que equivale a decir que el resultado es un múltiplo de 16 palabras de 32-bits. Podemos representar el mensaje como, M[0...N-1], con N un entero múltiple de 16. De esta manera, N = L·16.

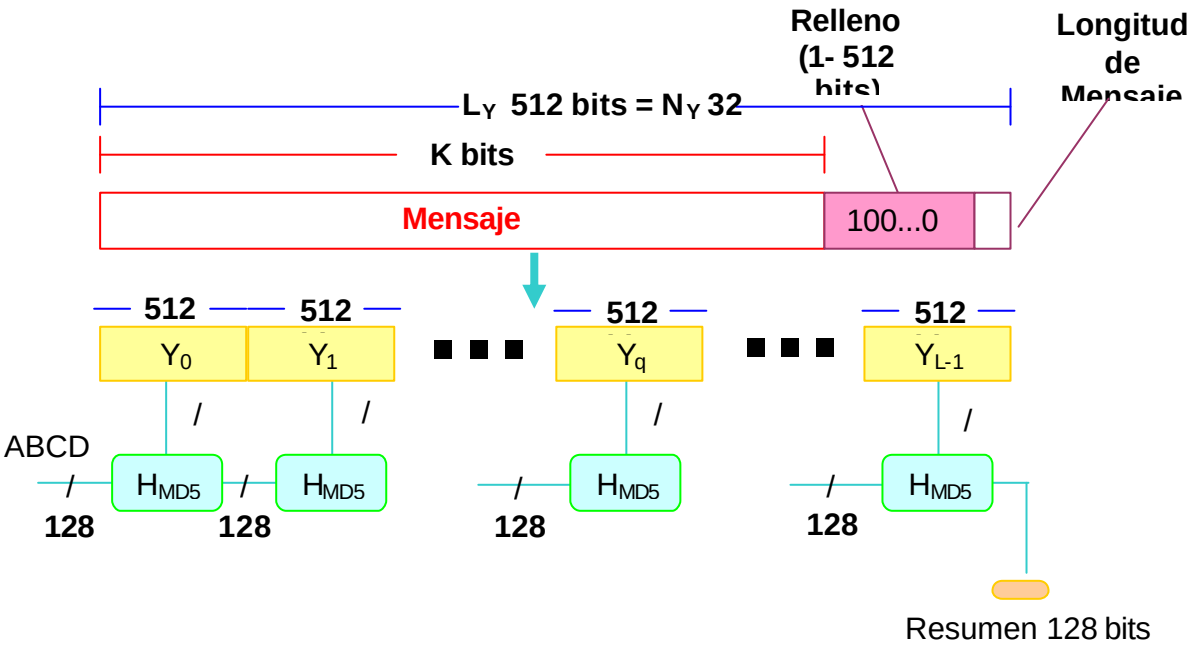


Figura 35 Generación de un Resumen de Mensaje Utilizando MD5

PASO 3. Inicializar el Buffer MD. Un buffer de 128-bits es utilizado para almacenar resultados intermedios y finales de la función HASH. El buffer puede ser representado como 4 registros de 32-bits (A, B, C, D). Estos registros son inicializados con los siguientes valores hexadecimales (los octetos menos significativos primero).

A = 01234567
B = 89abcdef
C = fedcba98
D = 76543210

PASO 4. Proceso de un mensaje en bloques de 512 bits (16 Palabras de 32 bits). **El corazón del algoritmo es un módulo que consiste en 4 ciclos de procesamiento. Este módulo es llamado como H_{MD5} en la *Figura 35*, y su lógica está ilustrada en la *Figura 36* Los 4 ciclos tienen una estructura similar, pero cada uno utiliza una función primitiva lógica diferente, referidas como *F, G, H* y *I*. En la *Figura 36*, los cuatro ciclos son etiquetados como f_F, f_G, f_H, f_I, para indicar que cada uno de los ciclos tiene en general la misma función estructurada, f, pero que depende de las diferentes funciones primitivas *(F,G,H,I)*.**

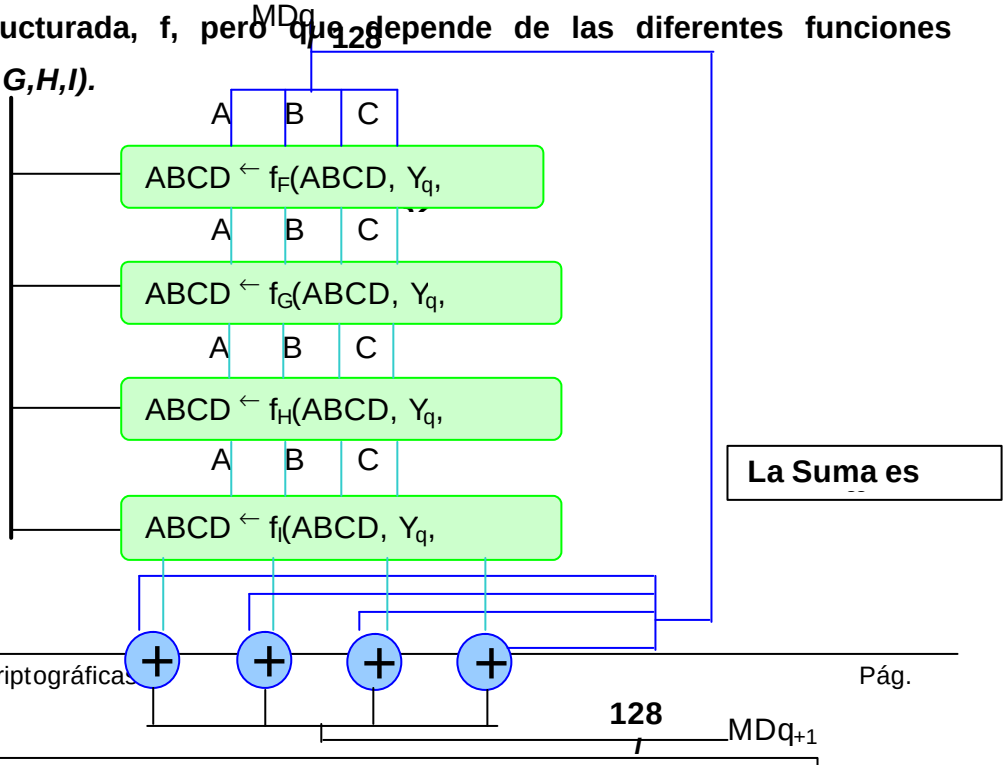


Figura 36 Actualización de un único bloque de 512-bits (H_{MD5})

Note que cada uno de los ciclos toma como entrada un bloque de 512 bits para ser procesado *(Y_q)* y el buffer de 128-bits con valor *ABCD*, y actualiza el contenido del buffer. Cada ciclo también utiliza 16 palabras de 32-bits de una tabla de 64 elementos, *T[1,...,64],* construidos a partir de una función Seno. El i-esimo elemento de *T*, denotado como *T[i]* tiene un valor igual a la parte entera de *(2^{32} x abs(sen(i)))*, donde *i* está en radianes. Ya que el *(abs(sen(i)))* es un número entre 0 y 1, cada elemento de *T* es un entero que puede ser representado en 32-bits. La tabla proporciona aleatoriamente palabras de 32-bits, con lo cual debería eliminar cualquier irregularidad en la entrada de datos. La *Tabla 14* lista los valores de *T*.

Tabla 14 Elementos de la Clave del MD5.

Tabla de verdad de las Funciones Lógicas

b	c	d	F	G	H	I
0	0	0	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	1
1	0	1	0	1	0	1

1 1 0 1 1 0
0
1 1 1 1 1 1
0

Tabla T, Construida a partir de la Función Seno.

T[1] = **D76AA478**	T[14] = **FD987193**	T[27] = **F4D50D87**	T[40] = **BEBFBC70**	T[53] = **655B59C3**
T[2] = **E8C7B756**	T[15] = **A679438E**	T[28] = **455A14ED**	T[41] = **289B7EC6**	T[54] = **8F0CCC92**
T[3] = **242070DB**	T[16] = **49B40821**	T[29] = **A9E3E905**	T[42] = **EAA127FA**	T[55] = **FFEFF47D**
T[4] = **C1BDCEEE**	T[17] = **F61E2562**	T[30] = **FCEFA3F8**	T[43] = **D4EF3085**	T[56] = **85845DD1**
T[5] = **F57C0FAF**	T[18] = **C040B340**	T[31] = **676F02D9**	T[44] = **04881D05**	T[57] = **6FA87E4F**
T[6] = **4787C62A**	T[19] = **265E5A51**	T[32] = **8D2A4C8A**	T[45] = **D9D4D039**	T[58] = **FE2CE6E0**
T[7] = **A8304613**	T[20] = **E9B6C7AA**	T[33] = **FFFA3942**	T[46] = **E6DB99E5**	T[59] = **A3014314**
T[8] = **FD469501**	T[21] = **D62F105D**	T[34] = **8771F681**	T[47] = **1FA27CF8**	T[60] = **4E0811A1**
T[9] = **698098D8**	T[22] = **02441453**	T[35] = **69D96122**	T[48] = **C4AC5665**	T[61] = **F7537E82**
T[10] = **8B44F7AF**	T[23] = **D8A1E681**	T[36] = **FDE5380C**	T[49] = **F4292244**	
T[11] = **FFFF5BB1**	T[24] = **E7D3FBC8**	T[37] = **A4BEEA44**	T[50] = **432AFF97**	
T[12] = **895CD7BE**	T[25] = **21E1CDE6**	T[38] = **4BDECFA9**	T[51] = **AB9423A7**	
T[13] = **6B901122**	T[26] = **C33707D6**	T[39] = **F6BB4B60**	T[52] = **FC93A039**	

Incluyendo todo, parar un bloque Y_q, el algoritmo toma Y_q y un código resumen intermedio con valor MD_q como entrada. MD_q es colocado en el buffer *ABCD*. La salida del cuarto ciclo es añadida a MD_q para crear MD_{q+1}. La suma está realizada independientemente de cada una de las cuatro palabras que hay en el buffer, con cada una de las correspondientes palabras que hay en MD_q, utilizando la suma módulo 2^{32}.

***PASO 5: Salida*: Después de que todos los L bloques de 512-bits hayan sido procesados, la salida de la L-ésima fase es un mensaje resumen de 128-bits. A continuación se estudia con más detalles la lógica de cada uno de los cuatro ciclos de procesamiento de un bloque de 512-bits. Cada ciclo consiste en una secuencia de 16 pasos los cuales operan sobre un buffer ABCD. Cada paso sigue el esquema ilustrado en la *Figura 37*.**

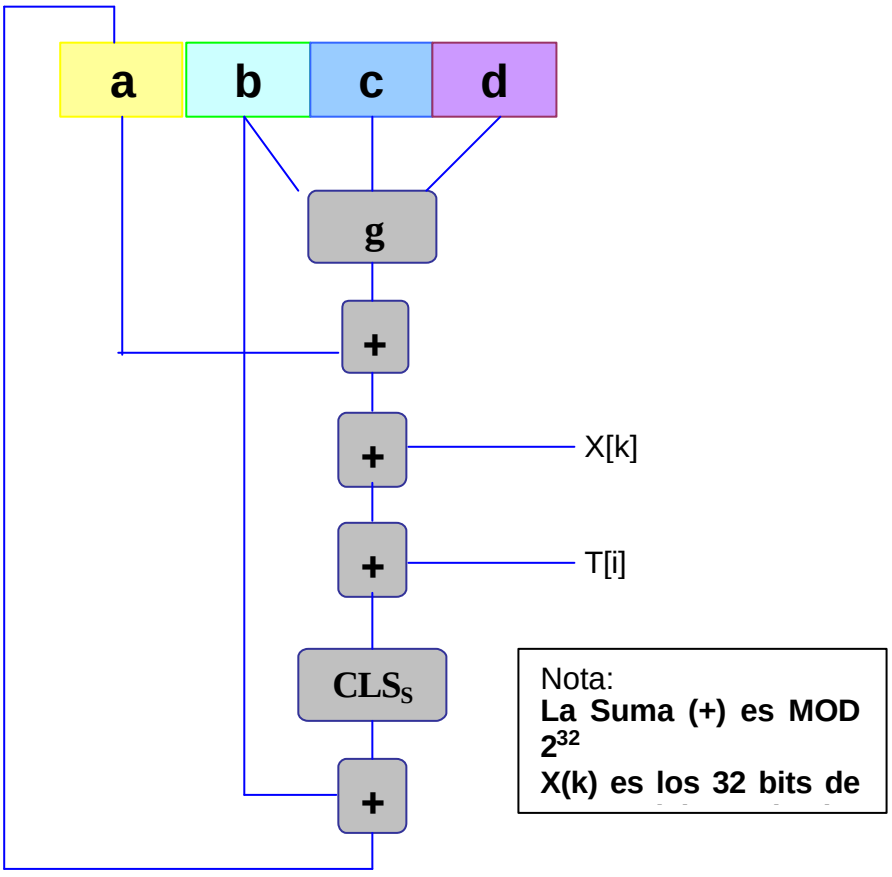


Figura 37 Operación elemental del MD5: (abcd k s i)

$$a \ddot{U} b + CLS_s (a + g(b, c, d) + X[k] + T[i])$$

Donde :

a,b,c,d **= Las cuatro palabras del buffer, en un orden que varia a través de los pasos.**

G **= Una de las funciones primitivas F, G, H, I.**

CLSs **= Desplazamiento circular a la izquierda de s bits sobre palabras de 32 - bits.**

X[k] **= M[qx16+k] = la k -ésima palabra de 32-bits en el q-ésimo bloque de 512-bits de un mensaje.**

T[i] **= La i-ésima palabra de 32-bits en un array T.**

\+ **= Suma mod 2^{32}.**

Una de las cuatro funciones lógicas primitivas es utilizada por cada uno de los cuatro ciclos del algoritmo. Cada función primitiva toma tres palabras de 32-bits como entrada y produce una palabra de 32-bits de salida. Cada función ejecuta una serie de operaciones lógicas; esto quiere decir, que el n-ésimo bit de salida es una función del n-ésimo bit de las tres entradas. Las funciones pueden ser resumidas tal y como se muestra a continuación:

Tabla 15 Funciones Lógicas del MD5

Ciclo	Función Primitiva g	g(b, c, d)
f_F	F(b, c, d)	(b · c) v (~b · d)
f_G	G(b, c, d)	(b · c) v (b · ~d)
f_H	H(b, c, d)	b ⊕ c ⊕ d
f_I	I(b, c, d)	C ⊕ (b · ~d)

Los operadores lógicos (AND, OR, NOT, XOR) son representados por los símbolos (·, v, ~,⊕). La función *F* es una función condicional (**sí** b **entonces** c **sino** d); Semejantemente *G* (**sí** d **entonces** b **sino** c); La función *H* produce un bit de paridad. La *tabla 14* es una tabla de la verdad de las cuatro funciones.

A continuación, adaptada de la *RFC 1321*, la definición del procedimiento del algoritmo del *Paso 4*. La expresión *(w <<< s)* denota el valor de 32-bits, obtenidos por la ejecución del desplazamiento circular a la izquierda (rotación) de *W* por *s* posiciones de bit. El array de palabras de 32-bits, X[0..15] , mantiene el valor actual del bloque de entrada, 512-bits, que serán procesados. Se ha de mencionar que cada ciclo consiste en 16 de un total de 64 pasos. Cada palabra de 32-bits de entrada es utilizada cuatro veces, una por ciclo, y cada una de las 64 palabras de 32-bits pertenecientes a *T* es utilizada exactamente una vez. También, anotar que por cada paso, sólo uno de los cuatro bytes del buffer *ABCD* es actualizado. De aquí en adelante, cada byte del buffer es actualizado 16 veces durante el ciclo, y entonces al decimoséptimo paso, el final, produce la salida final del bloque.

```
Procesa bloques de 16 palabras de 32 bits (512 bits).

For i = 0 to N/16-1 do
  /* Copiar bloque i en X. */
For j = 0 to 15 do
   Copiar en X[j] => M[i*16+j].
End
/* Guardar A como AA, B como BB, C como CC,
                    y D como DD. */
                        AA = A
                        BB = B
                        CC = C
                        DD = D
/* Ciclo Nº 1. */
/* Con [abcd k s i] realiza la operación a = b +
((a + F(b,c,d) + X[k] + T[i]) <<< s). */
/* Hace las siguientes 16 operaciones. */
                    [ABCD  0   7  1]
                    [DABC  1  12  2]
                    [CDAB  2  17  3]
                    [BCDA  3  22  4]
                    [ABCD  4   7  5]
                    [DABC  5  12  6]
                    [CDAB  6  17  7]
```

```
/* Ciclo Nº 2. */
/* Con [abcd k s i] realiza la operación a = b + ((a +
G(b,c,d) +
X[k] + T[i]) <<< s). */
/* Hace las siguientes 16 operaciones. */

                    [ABCD  1   5 17]
                    [DABC  6   9 18]
                    [CDAB 11 14 19]
                    [BCDA  0 20 20]
                    [ABCD  5   5 21]
                    [DABC 10   9 22]
                    [CDAB 15 14 23]
                    [BCDA  4 20 24]
                    [ABCD  9   5 25]
                    [DABC 14   9 26]
                    [CDAB  3 14 27]
                    [BCDA  8 20 28]
                    [ABCD 13   5 29]
                    [DABC  2   9 30]
                    [CDAB  7 14 31]
                    [BCDA 12 20 32]
```

[BCDA 7 22 8] **[ABCD 8 7 9]** **[DABC 9 12 10]** **[CDAB 10 17 11]** **[BCDA 11 22 12]** **[ABCD 12 7 13]** **[DABC 13 12 14]** **[CDAB 14 17 15]** **[BCDA 15 22 16]**	
/* Ciclo Nº 3. */ **/* Con [abcd k s t] realiza la operación a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */** **/* Hace las siguientes 16 operaciones. */** **[ABCD 5 4 33]** **[DABC 8 11 34]** **[CDAB 11 16 35]** **[BCDA 14 23 36]** **[ABCD 1 4 37]** **[DABC 4 11 38]** **[CDAB 7 16 39]** **[BCDA 10 23 40]** **[ABCD 13 4 41]** **[DABC 0 11 42]** **[CDAB 3 16 43]** **[BCDA 6 23 44]** **[ABCD 9 4 45]** **[DABC 12 11 46]** **[CDAB 15 16 47]** **[BCDA 2 23 48]**	/* Ciclo Nº4 . */ **/* Con [abcd k s t] realiza la operación a = b + ((a + I(b,c,d) +** **X[k] + T[i]) <<< s). */** **/* Hace las siguientes 16 operaciones. */** **[ABCD 0 6 49]** **[DABC 7 10 50]** **[CDAB 14 15 51]** **[BCDA 5 21 52]** **[ABCD 12 6 53]** **[DABC 3 10 54]** **[CDAB 10 15 55]** **[BCDA 1 21 56]** **[ABCD 8 6 57]** **[DABC 15 10 58]** **[CDAB 6 15 59]** [BCDA 13 21 60] [ABCD 4 6 61] **[DABC 11 10 62]** **[CDAB 2 15 63]** **[BCDA 9 21 64]** **/* Incrementa cada uno de los cuatro registros con el valor que** **tenían antes de empezar el proceso.** **A = A + AA** **B = B + BB** **C = C + CC** **D = D + DD** **end /* final del ciclo nº i */**

Figura 38 Funcionamiento básico del Algoritmo MD5. (Según RFC 1321)

El comportamiento anterior del MD5 se resume como sigue:

$$MD_0 = IV$$

$$MD_{q+1} = MD_q + f_I [Y_q, f_H [Y_q, f_G [Y_q, f_F [Y_q, MD_q]]]]$$

$$MD = MD_{L-1}$$

Donde :

IV = Valor inicial del buffer ABCD, definido en el *Paso3*.

Y_q = El q-ésimo bloque de 512-bits del mensaje.

L = El número de bloques en el mensaje (Incluyendo el relleno y el campo longitud).

MD = Valor final de resumen del mensaje.

4.1.1.2 FORTALEZA DEL MD5

El algoritmo *MD5* tiene la propiedad de que cada bit del código Hash es una función de cada bit de entrada. La compleja repetición de las funciones básicas (*F, G, H, I*) produce unos resultados que están bien mezclados; siendo desfavorable que dos mensajes elegidos aleatoriamente, aunque demuestren similares regularidades, tengan el mismo código Hash. Rivest conjetura en el *RFC* que el *MD5* es tan fuerte como es posibles para un código Hash de 128-bits, es decir, la probabilidad de obtener dos mensaje con el mismo resumen es de 2^{64} operaciones, mientras que la dificultad de encontrar un mensaje a partir de un resumen dado se calcula en alrededor de 2^{128} operaciones.

4.1.2 CONSIDERACIONES DE SEGURIDAD

Los usuarios deben entender que la calidad de la seguridad provista por esta especificación depende completamente del algoritmo MD5: que este bien implementado, la

seguridad del manejo de la llave y su implementación y sobre todo que sea correctamente aplicado en todos los nodos participantes.

Se sabe que es posible que se produzcan colisiones en la función de compresión de *MD5.* No hay en la práctica un método conocido para explotar estas colisiones con el fin de atacar *MD5.* Ha sido determinado recientemente que es posible construir una máquina por $10 millones de dólares que pueda encontrar 2 variantes de texto para un código *MD5* dado. De todas maneras no esta claro si este ataque es aplicable a una transformación *MD5* con llave (es decir nunca se llevo a la practica).

Este ataque requeriría unos 24 días para llegar a su cometido. La misma forma de ataque es la comúnmente usada en cualquier función iterativa de n bits, es decir explorar todas las posibles combinaciones, prueba y error, el tiempo se relaciona con los 128 bits de *MD5* a causa de la inmensa cantidad de configuraciones posibles.

4.1.2.1 CLAVES

La clave de autenticación que se comparte en forma secreta entre las partes que se comunican debe ser un numero aleatorio fuertemente encriptado y no una cadena de ningún tipo. No se obliga a la llave encriptada por esta transformación a tener algún tamaño en especial pero su longitud es de hasta 128 bits y debe ser soportada por la implementación, sin embargo alguna llave en particular puede ser más corta.

4.1.3 IMPLEMENTACION

Se puede realizar de muchas maneras, una de ellas es mediante una función, la cual recibe los siguientes parámetros:

Figura 39 Implementación Función de Resumen MD5

Texto Plano o Resumen, **dependiendo de lo que se desee la función recibe ya sea le texto plano o el resumen del mismo, de igual forma devuelve lo contrario de lo que recibe como se puede notar en la gráfica.**

4.2 SHA

El algoritmo *SHA* fue desarrollado por la *NSA*, para ser incluido en el estándar *DSS(Digital Signature Standard*). Al contrario que los algoritmos de cifrado propuestos por esta organización, *SHA* se considera seguro y libre de puertas traseras, ya que favorece a los propios intereses de la *NSA* que el algoritmo sea totalmente seguro.

SHA puede ser usado para generar el resumen de un mensaje, es decir, una versión condensada del mensaje original. El uso es apropiado cuando se utilizan firmas digitales, que pueden ser utilizadas en correo electrónico, transferencias bancarias, distribución de software, almacenaje de datos y otras aplicaciones que requieran integridad de los datos y autenticación del origen de los datos. El SHA también puede ser usado donde sea necesario generar una versión reducida de un mensaje. Este algoritmo puede ser implementado tanto en software, como en hardware o en combinación de ambos.

4.2.1 DESCRIPCION

4.2.1.1 LOGICA DEL SHA

El algoritmo toma como entrada un mensaje de longitud máxima 2^{64} bits y produce como salida un resumen del mensaje de 160 bits. La entrada es procesada en bloques de 512-bits. Este algoritmo esta conformado por los siguientes pasos.

PASO 1. Añadir relleno de bits. **El mensaje es rellenado para que la longitud en bits sea congruente con 448 módulo 512 (longitud ° 448 mod 512). El relleno es añadido siempre, aunque el mensaje ya sea de la longitud deseada. Por lo tanto, el número de bits de relleno se sitúa entre 1 y 512 bits. El relleno consiste en un único bit 1 seguido del número necesario de bits 0.**

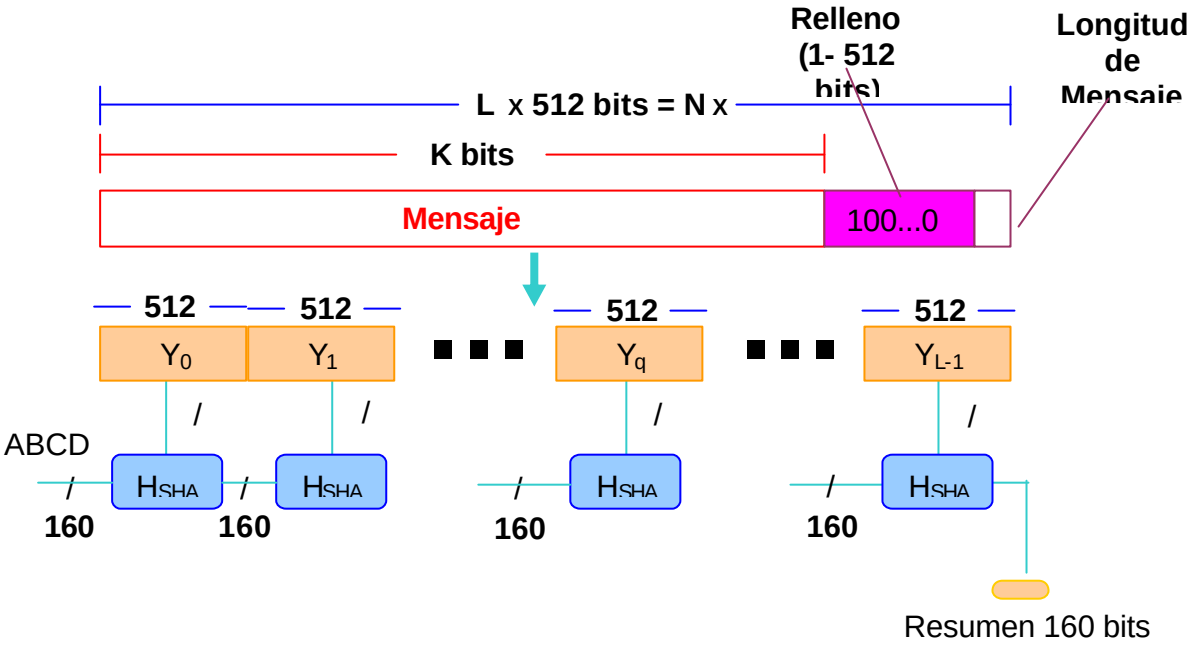


Figura 40 Generación de un Resumen de Mensaje Utilizando SHA

PASO 2. Añadir longitud. Una bloque de 64 bits es añadido al mensaje. Este bloque es tratado como un entero de 64 bits sin signo y contiene la longitud del mensaje original(antes del relleno).

El resultado de los dos primeros pasos produce un mensaje de longitud múltiplo de 512 bits(numero entero). En la *Figura 40,* el mensaje extendido es representado como una secuencia de bloques de 512 bits $Y_0, Y_1, ..., Y_{L-1}$, por lo que la longitud total del mensaje extendido es Lx512 bits. Lo que equivale a decir que el resultado es un múltiplo de 16

palabras de 32-bits. Podemos representar el mensaje como, M[0...N-1], con N un entero múltiple de 16. De esta manera, N = Lx16.

PASO 3. Inicializar el Buffer MD. Un buffer de 160-bits es utilizado para almacenar resultados intermedios y finales de la función HASH. El buffer puede ser representado como 5 registros de 32-bits (A, B, C, D, E). Estos registros son inicializados con los siguientes valores hexadecimales (La parte alta del primer octeto).

A = 67452301
B = EFCDAB89
C = 98BADCFE
D = 10325476
E = C3D2E1F0

Observe que los 4 primeros valores son los mismo que los usados en el MD5; la diferencia en la apariencia es causada por el uso de una convención diferente para expresar los valores(Parte alta Vs Parte baja de primer octeto)

PASO 4. Proceso de un mensaje en bloques de 512 bits (16 Palabras de 32 bits). **El corazón del algoritmo es un módulo que consiste en 80 ciclos de procesamiento. Este módulo es llamado como H_{SHA} en la *Figura 40*, y su lógica está ilustrada en la *Figura 41*. Los 80 ciclos tienen una estructura similar, observé que cada paso toma como entrada el bloque actual de 512 bits que esta siendo procesado (Y_q) y el valor ABCDE del buffer de 160 bits, actualizando el contenido del buffer. Cada paso hace uso de una constante aditiva Kt, en si únicamente 4 diferentes constantes son usadas. Los valores, en el sistema Hexadecimal son los siguientes:**

Tabla 16 Valores de las Constantes Usadas SHA		
K(t) = 5A827999		(0 <= t <= 19)
K(t) = 6ED9EBA1		(20 <= t <= 39)
K(t) = 8F1BBCDC		(40 <= t <= 59)
K(t) = CA62C1D6		(60 <= t <= 79)

En general para un bloque Yq, el algoritmo toma Yq y un valor intermedio MDq como entradas. MDq es colocado dentro de buffer ABCDE . La salida del ciclo 80 es agregada a MDq para producir MDq+1. La suma es echa independientemente para cada una de la cinco palabras en el buffer con cada una de las correspondientes palabras en MDq, usando suma MOD 2^{32}.

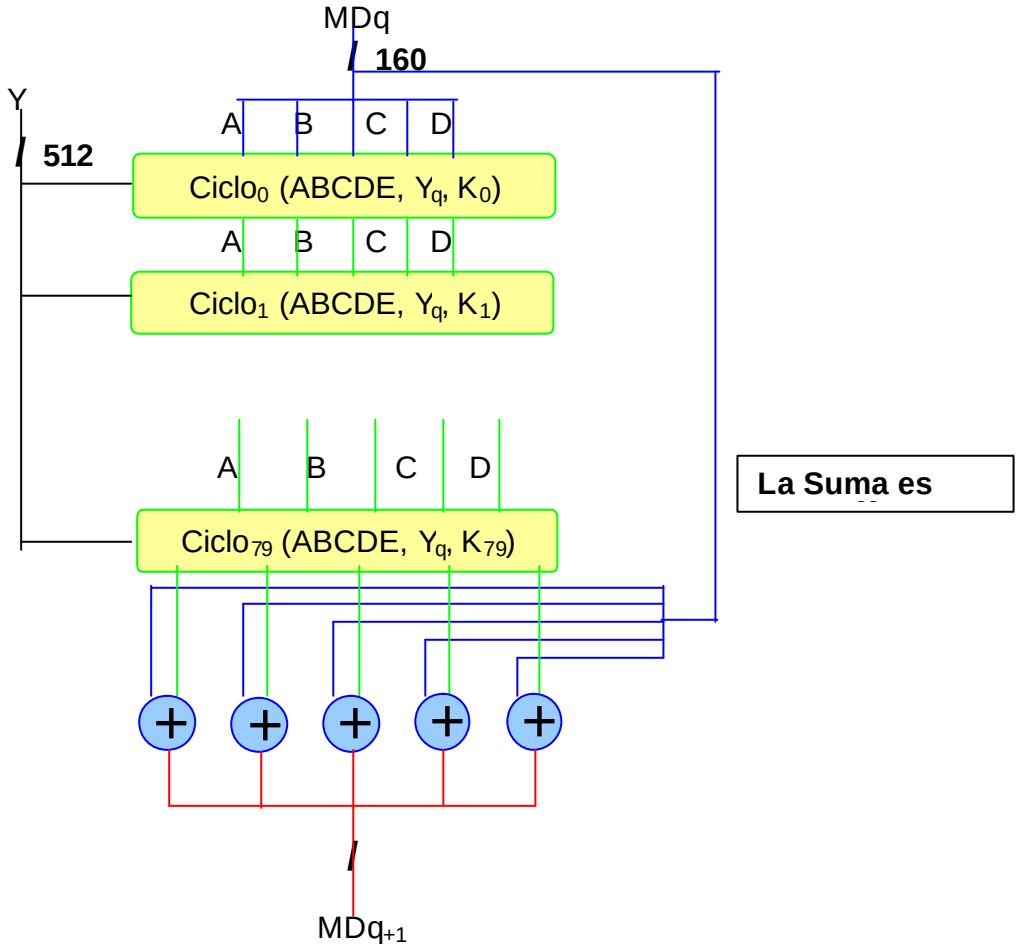


Figura 41 Actualización de un único bloque de 512-bits (H_{SHA})

***PASO 5: Salida*: Después de que todos los L bloques de 512-bits han sido procesados, la salida de la L-ésima fase es un mensaje resumen de 160-bits. A continuación se estudia con más detalles la lógica de cada uno de los 80 pasos o ciclos de procesamiento de un bloque de 512-bits. Cada paso sigue el esquema ilustrado en la *Figura 42*.**

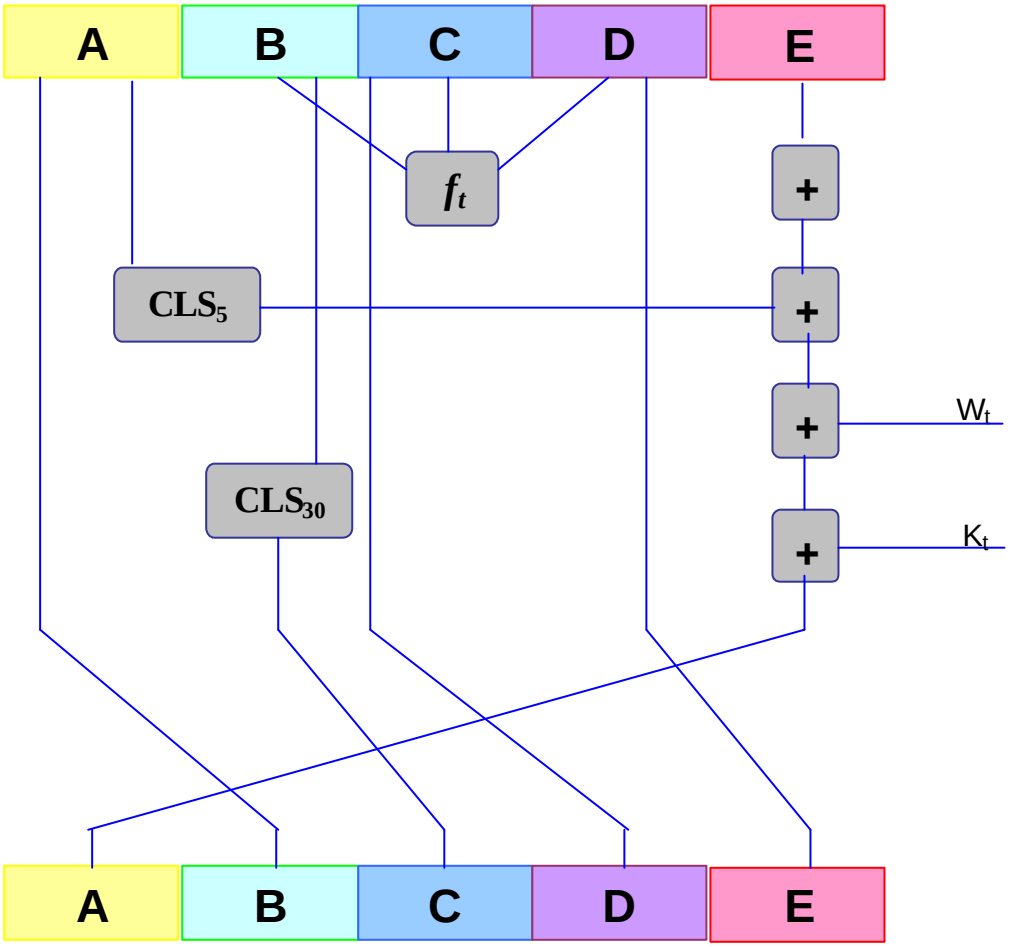


Figura 42 Operación elemental del SHA

$$A, B, C, D, E \ddot{U} (CLS_5(A) + f_t(B, C, D) + E + W_t + K_t), A, CLS_{30}(B), C, D$$

Donde :

A,B,C,D,E **= Las cinco palabras del buffer.**

t **= Numero del ciclo o paso que se esta efectuando(0 £ t £ 79).**

f_t **= Función Lógica Primitiva.**

CLSs **= Desplazamiento circular a la izquierda de s bits sobre palabras de 32 - bits.**

W_t **= palabra de 32-bits deriva del bloque de entrada actual 512-bit**

K_t **= Constante aditiva, 4 diferentes valores son usados como se definieron anteriormente.**

\+ **= Suma mod 2^{32}.**

Cada función primitiva toma tres palabras de 32-bits como entrada y produce una palabra de 32-bits de salida. Cada función ejecuta una serie de operaciones lógicas; esto quiere decir, que el n-ésimo bit de salida es una función del n-ésimo bit de las tres entradas. Las funciones pueden ser resumidas tal y como se muestra a continuación:

Tabla 17 Operaciones Lógicas del SHA

$f_t(B, C, D)$	Ciclo
$(B \bullet C) \vee (\overline{B} \bullet D)$	(0<= t <= 19)
$B \oplus C \oplus D$	(20<= t <= 39)
$(B \bullet C) \vee (B \bullet D) \vee (C \bullet D)$	(40<= t <= 59)
$B \oplus C \oplus D$	(60<= t <= 79)

Como puede ser visto únicamente 3 diferentes funciones son usadas. Para los ciclos de 0 -19 la función es condicional : (sí **B** entonces **C** sino **D); para los ciclos 20 –39 y 60-79, la función produce un Bit de paridad. Para los ciclos 40-59 , la función es verdadera si dos o tres de los argumentos son verdaderos en la tabla 18 es una tabla de verdad de estas funciones.**

Tabla 18 Tabla de Verdad de Funciones Lógicas del SHA						
B	c	D	$f_{0..19}$	$f_{20..39}$	$F_{40..59}$	$F_{60..79}$
0	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**1**	**1**	**1**	**0**	**1**
0	**1**	**0**	**0**	**1**	**0**	**1**
0	**1**	**1**	**1**	**0**	**1**	**0**
1	**0**	**0**	**0**	**1**	**0**	**1**
1	**0**	**1**	**0**	**0**	**1**	**0**
1	**1**	**0**	**1**	**0**	**1**	**0**
1	**1**	**1**	**1**	**1**	**1**	**1**

4.2.2 IMPLEMENTACION

Se puede realizar de muchas maneras, una de ellas es mediante una función, la cual recibe los siguientes parámetros:

Figura 43 Implementación Función de Resumen SHA

Texto Plano o Resumen, **dependiendo de lo que se desee la función recibe ya sea le texto plano o el resumen del mismo, de igual forma devuelve lo contrario de lo que recibe como se puede notar en la gráfica.**

5. CONCLUSIONES Y RECOMENDACIONES

5.1 CONCLUSIONES

5.1.1 DE LA IMPLEMENTACION DE LOS ALGORITMOS

La implementación se desarrolló para que estos cifradores y funciones HASH puedan ser usados con cualquier tipo de archivo.

Las librerías desarrolladas en C++ son 100 % compatibles con C-ANSI, para esto se desarrollaron funciones de manejo de cadenas (principalmente) ya que las existentes en las librerías de C++ no estaban declaradas en el estándar C-ANSI.

Se debió desarrollar dos librerías adicionales en los diferentes lenguajes las cuales se clasifican como utilitarias y que además de servir para nuestro propósito, pueden ser de gran ayuda para estudiantes o profesores que estén desarrollando alguna aplicación que requiera de estas librerías o desarrolladores en estos lenguajes independientes del mundo de la criptográfica.

La primera librería utilitaria (llamada *Archivo_*) es la de manejo de archivos y otras funciones entre las que se destacan concatenado, separado, comparación, tamaño, borrado, creado entre otras. Además de lo anterior, también cuenta con funciones para el manejo de cadenas. Esta librería es

utilizada por todas las librerías de los Cifradores y funciones HASH. EL contenido de la misma librería se encuentra explicado en el Manual de Mantenimiento de las Herramientas Criptográficas (Anexo A).

La segunda librería Utilitaria (Llamada Nlargos_) es para el manejo de números grandes, debido a que los lenguajes de programación como el C y el Pascal no cuentan con ningún tipo de variable o funciones que puedan operar y almacenar números de mas de 100 dígitos, siendo esta una condición indispensable para el desarrollo de los Cifradores LUC y RSA cuya principal característica es el manejo de números grandes. Es así como se implementa una librería para cada lenguaje tratado, en las que se manejan las operaciones básicas (Suma, Resta, Multiplicación, Modulo y Parte entera del Cociente) con la capacidad de manejar hasta 2^{32} (65535) dígitos. Para la implementación de la misma se utilizó una gran destreza en el manejo de la memoria, valiéndonos de técnicas de administración de memoria, basados en los amplios conocimientos que se tiene de los lenguajes de programación sugeridos. Lamentablemente se necesita mucha velocidad en el proceso de operaciones simultaneas para los procesos internos de los cifradores LUC y RSA, para los cuales la librería es lenta para dicha tarea por estar programada a alto nivel. En el CD hay una pequeña aplicación que sirve para ayudar a evaluar esta librería la cual se encuentra en el directorio d:\Calculadora.

Sobre el desarrollo de las librerías referentes a los cifradores de transposición Julio cesar, Monoalfabetico y Vigenère no hubo ningún problema, puesto que su implementación es relativamente fácil y son un excelente material de apoyo para entender los objetivos y funcionamiento de la criptográfica, siendo útil para la materia de Criptografía y Seguridad de Redes y en general a cualquier persona interesada en el tema. Algo interesante sobre estos cifradores es que están implementados para que

funcionen con cualquier tipo de archivo, desmintiendo así la creencia de que solo sirve para archivos texto.

Para el DES e IDEA se implementaron de forma muy similar, hay referencias bibliográficas del triple y doble DES, en base a estas se decidió implementar estas funciones para el cifrador IDEA creyendo que no exponen la seguridad del algoritmo, basándose en consultas y experiencias que se realizaron con el DES en el transcurso de este proyecto.

Otra variante que se adaptó para darle fortaleza al algoritmo es la posibilidad que puede tener el usuario de variar Jbits en el rango de 1 a 64 bits en los modos de operación CFB y OFB, rompiendo con el estándar de 8 bits, dejando a voluntad del usuario su escogencia, la cual implica un reto más para un atacante. Entre menor sea el valor de Jbits mayor es la seguridad brindada por el algoritmo a costa de un mayor tiempo en el proceso de cifrado.

Esta implementación de Jbits variable conlleva a un problema, como es sabido que la mínima unidad que se puede procesar en un computadora es de 8 bits (1 Byte) entonces si el valor de jbits es menor o no es múltiplo de 8, quedaran bits sin procesar del archivo de entrada, la solución de este problema se expone en el Manual de Mantenimiento de las Librerías Criptográficas (Anexo A).

Para trabajar cualquier archivo con los cifradores LUC y RSA se debió implementar una función para convertir el archivo que se quiere cifrar a formato numérico en base 10, de la siguiente manera: El archivo leído tiene una representación numérica en base 256 (Formato ASCII) la cual se cambia a una representación numérica base 10, siendo este último el texto que se le da a los

cifradores (LUC, RSA) como parámetro de entrada. El valor devuelto por estos dos cifradores se encuentra en base 10. Para almacenar este valor devuelto en disco se debe cambiar su representación en base 256 (Formato ASCII). Todo esto se debe a que el LUC y el RSA trabajan con operaciones matemáticas y trata a los archivos como números.

El desarrollo de estos cifradores se limita a las funciones de Encripción y Desencripción de archivos, nunca fue objetivo de esta investigación la generación de claves para estos cifradores, además es conocido que el número de operaciones y la longitud de los números que maneja es grande haciendo al cifrador bastante lento.

Se consideró como desventaja del LUC el proceso de Desencripción debido a que para cada Clave Pública hay 4 posibles Claves Privadas. Para desencriptar un archivo cualquiera encriptado con la clave pública, el usuario tendría que probar las 4 claves privadas, y luego ver cual de los 4 archivos desencriptados es el que corresponde al archivo original; este mecanismo solo funciona si el archivo a Encriptar contiene información con sentido (como una frase, un texto, etc), en caso que el archivo contuviera información sin sentido (como puede ser una clave, una serie de bits) el usuario no tendría manera de saber cual de los 4 archivos desencriptados es el verdadero. La solución para este problema es utilizar una función llamada Jacobi la cual a partir de las 4 claves privadas, los valores de *p* y *q*, y el archivo a Desencriptar calcula cual de las 4 clave sirve para Desencriptar este archivo. El inconveniente que observamos es que esta función requiere los valores *p* y *q*, violando así uno de los principales requisitos de un algoritmo de clave pública el cual es que no se conozcan estos valores, puesto que se expone al generador de claves.

En la implementación de la función de Encripción y Desencripción de LUC no se tuvo en cuenta este problema debido a que la función Jacobi corresponde a un módulo aparte que debe ser llamado antes de la Desencripción, y la implementación de dicho módulo no esta definido dentro de los alcances de este proyecto.

En el caso de las funciones HASH , particularmente el MD5 se observó que en la parte de los 64 pasos que están divididos en 4 ciclos de 16 pasos cada uno, estos seguían cierto comportamiento el cual podía ser programado en una pequeña rutina, ahorrándose así la necesidad de escribir 64 líneas diferentes de código. Este caso también se presentó para el SHA el cual con una pequeña rutina de pocas líneas de código se evitó escribir 80 líneas de código. Para este mismo algoritmo se creó un mecanismo recursivo con el fin de obtener del buffer de 512 bits a Wt (Ver código fuente librería Md5_ y Sha_ que se encuentran en el CD).

5.1.2 DE LA IMPLEMENTACION DE LA APLICACIÓN

EL propósito de la implementación de la aplicación es el de facilitar la evaluación de las librerías y que además esta sirva como material ilustrativo y de apoyo a las asignaturas que hagan referencia a la criptografía. Con este propósito se desarrolló una aplicación bastante amigable y fácil de usar para personas con un mínimo de conocimiento de criptografía, aunque en caso de no saber nada sobre el tema, la aplicación cuenta con una excelente ayuda la cual le servirá para incursionar en este nuevo mundo criptográfico.

La aplicación cuenta con todos los cifradores desarrollados y funciones HASH, además de tener la ventaja de brindarle al usuario la posibilidad de utilizar algunos servicios como son la autenticación y la firma digital. Otra cualidad es que la aplicación dispone de otras herramientas como son concatenar, separar y comparar archivos las cuales les pueden ser de mucha utilidad al usuario en caso de que desee utilizar algún esquema de servicio con el cual no cuenta la aplicación pero que se muestra en la ayuda o simplemente si se idea uno. Por otra parte la aplicación cuenta con mecanismos visuales que guían al usuario en la comprensión de la misma, estos mecanismos son la ayuda interactiva disponible, como son los mensajes que aparecen cada vez que el puntero del ratón se ubica sobre alguna opción o área de trabajo específica.

La programación visual de los cifradores de sustitución, siendo el caso especifico el Monoalfabetico fue las mas complicada en cuestión de diseño, debido a la alta interacción que debe hacer el usuario en esta función, como es la escogencia de una Alfabeto sustituto el cual es un proceso tedioso para el usuario, creemos que el diseño de la ventana de este cifrador es una de las formas mas amigables y fáciles para el usuario.

Refiriéndonos a la funcionalidad de la aplicación, consideramos que un usuario común le podrá sacar mucho provecho, exceptuando las funciones de LUC y RSA para claves de mas de 30 dígitos.

El desarrollo y diseño de esta aplicación se basa en mecanismos heurísticos, los cuales facilitan la interacción con el usuario, además cumple con la mayoría de los requisitos de Windows para contar con su Logotipo.

5.1.3 CONCLUSIONES GENERALES

El desempeño de los algoritmos depende mucho de la máquina en que se este trabajando, sobre todo el RSA y LUC debido a que las operaciones matemáticas exigen bastantes recursos de ésta en el procesamiento.

Posteriormente se seguirá trabajando en la aplicación, ya que hacen falta las opciones de generación de claves para el LUC y RSA, además del mejoramiento de desempeño de la librería de manejo de números grandes, las cuales no se llevaron acabo por disponibilidad de tiempo y de no hacer parte de esta investigación

Se realizo una pagina de Internet la cual sirve de complemento de ayuda de la aplicación, brindando información del funcionamiento interno de los cifradores y funciones HASH para los usuarios que quieran profundizar sobre el tema, también contiene información sobre la aplicación, como sus características e instalador comprimido para que cualquier visitante pueda tener la aplicación. La dirección de la pagina es la siguiente http://www.gratisweb.com/wincrip2000 . La pagina cuenta con un enlace a la pagina de la universidad, no se monto directamente en el servidor de la misma por problemas de disponibilidad de tiempo para tramitar los permisos necesarios para dicho propósito.

5.2 RECOMENDACIONES

Se propone como trabajo de grado en el futuro la implementación de una librería de manejo de números grandes a bajo nivel(Lenguaje Maquina o directamente utilizando las librerías de Windows), la cual optimice las librerías de los cifradores de clave publicas expuestas en este documento.

Tener en cuenta las limitaciones de los lenguajes en que se implementaron las librerías, en cuanto al manejo de memoria y el manejo de cadenas (C-ANSI) en las cuales se debió implementar algunas funciones.

Promover la utilización de las librerías, para el desarrollo de aplicaciones para Internet, Intranet y protocolos seguros en próximos proyectos.

BIBLIOGRAFIA

OSIER, Dan, GROBMAN, Steve y BATSON, Steve. Aprendiendo DELPHI 2 en 21 días. Prentice-Hall Hispanoamericana, S.A. 1996.

CHAPMAN, Davis. DELPHI 2 Aplicaciones para Internet. Prentice-Hall, Madrid, 1997.

SCHILDT, Herbert. Turbo C/C ++ 3.1 Manual de Referencia. Segunda Edición. Osborne/McGraw-Hill, 1994.

STALLINGS, William. Network and Internetwork security. Principles and Practice. Prentice-Hall.

KAUFMAN, Charlie, PERLMAN, Radia y SPENICER, Mike. Network security.

ROBLINO, Dorothy. Cryptography and data secunty. AddisonWesley: Canada,1982.

KONUE@ Alan G.. Criptography: A Primer. John Wiley & Sons, Inc. Estados Unidos. 1981.

JOYANES AGUILAR, Luis. Programación en TURBO PASCAL versiones 5.5, 6.0 y 7.0. Segunda Edición. McGraw-Hill, 1993.

ARCILA, Jaime. Manual de Criptografía y Seguridad en Redes. 1998