INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INNOVACIÓN Y DESARROLLO TECNOLÓGICO EN CÓMPUTO

ROMPIMIENTO DEL ALGORITMO DES UTILIZANDO FUERZA BRUTA Y PROGRAMACIÓN PARALELA

T E S I S

Que para obtener el grado de

MAESTRÍA EN TECNOLOGÍA DE CÓMPUTO

PRESENTA

ING. FABIOLA ELIZABETH AGUILAR ESTRADA

DIRECTORES DE TESIS

M. EN C. JESÚS ANTONIO ÁLVAREZ CEDILLO

DR. VÍCTOR MANUEL SILVA GARCÍA

Noviembre 2011.

SIP-14-BIS

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México, D.F. siendo las 11:30 horas del día 22 del mes de Noviembre del 2011 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del CIDETEC para examinar la tesis titulada:

"ROMPIMIENTO DEL ALGORITMO DES UTILIZANDO FUERZA BRUTA Y PROGRAMACIÓN PARALELA"

Presentada por el alumno:

AGUILAR	ESTRADA	FABIOLA ELIZABETH
Apellido paterno	Apellido materno	Nombre(s)

Con registro:

B	0	9	1	3	4	6

aspirante de:

Maestría en Tecnología de Cómputo

Después de intercambiar opiniones los miembros de la Comisión manifestaron ***APROBAR LA TESIS***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA
Directores de tesis

M. EN C. JESÚS ANTONIO ÁLVAREZ CEDILLO
Primer Vocal

DR. VÍCTOR MANUEL SILVA GARCÍA
Segundo Vocal

DR. ROLANDO FLORES CARAPIA
Presidente

M. EN C. ISRAEL RIVERA ZÁRATE
Secretario

DR. JUAN CARLOS HERRERA LOZADA
Tercer Vocal

PRESIDENTE DEL COLEGIO DE PROFESORES

DR. VÍCTOR MANUEL SILVA GARCÍA

S.E.P.
INSTITUTO POLITECNICO NACIONAL
CENTRO DE INNOVACION Y DESARROLLO TECNOLOGICO EN COMPUTO

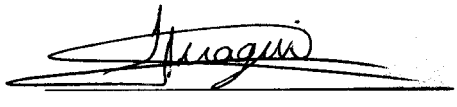
INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de México, D.F. el día 5 del mes Diciembre del año 2011, la que suscribe Fabiola Elizabeth Aguilar Estrada alumna del CIDETEC con número de registro B091346, adscrito al Programa de Maestría en Tecnología de Computo, manifiesta que es autora intelectual del presente trabajo de Tesis bajo la dirección de M. en C. Jesús Antonio Álvarez Cedillo y Dr. Víctor Manuel Silva García y cede los derechos del trabajo intitulado "Rompimiento del algoritmo DES utilizando fuerza bruta y programación paralela", al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección fabis104@hotmail.com. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.



Fabiola Elizabeth Aguilar Estrada

Resumen

El ocultamiento de la información a personas ajenas a la empresa ha sido siempre importante. Con la aparición de las computadoras fue necesario crear nuevos y mejores métodos de protección, dando como resultado la aparición de algoritmos de cifrado como el Estándar de Encriptación de Datos (DES por sus siglas en inglés). DES consta de 16 rondas, durante las cuales se hacen operaciones de sustitución, permutación y operaciones X-or con los 64 bits del mensaje y la llave elegida para el cifrado.

DES ha sido víctima de varios y variados ataques, entre los más conocidos tenemos el Criptoanálisis Diferencial, el Criptoanálisis Lineal y múltiples ataques de fuerza bruta. Los ataques de fuerza bruta realizados, generalmente requieren de una máquina construida con el único propósito de probar todas las posibles llaves o de utilizar cómputo distribuido.

En el presente trabajo, se plantea realizar un ataque por fuerza bruta al algoritmo DES, por medio de software, utilizando el cómputo paralelo con los procesadores de múltiples núcleos. Se creó un programa secuencial eficiente, que posteriormente fue paralelizado usando OpenMP, el cual añade concurrencia a los programas escritos en C. Con lo que es posible probar varias llaves a la vez dependiendo del número de núcleos en el procesador.

Después de probar el algoritmo, se pudo determinar que la aceleración del programa paralelo con respecto al programa secuencial es aproximadamente de 3.7.

A pesar de la disminución del tiempo de ejecución, al realizar el ataque utilizando los cuatro procesadores el tiempo de procesamiento es muy alto (más de 100 mil años) al probar todo el universo de llaves, por lo cual no es factible realizar este ataque utilizando este número de procesadores.

Abstract

To hide information from people outside the enterprise has been always important. When the computers showed up, it was necessary to create new and better protection methodologies, having as a result the creation of some encryption algorithms as the one known as DES (Data Encryption Standard). DES is an algorithm, consisting on 16 rounds. In each one, there are substitution, permutation and X-or operations with the 64 bits of the original message and the chosen key for the encryption.

DES has been attacked several times, among the most known attacks, we have the Differential Cryptanalysis, the Lineal Cryptanalysis and diverse brute force attacks. The brute force attacks done to DES, usually require a specific purpose computer, which has the objective of trying all the possible keys; they can also been done with distributed computing.

In this paper, a brute force attack to DES by software is proposed, using parallel computing with the multiple cores processors. In order to carry out the attack, it was necessary to create an efficient sequential program, that later was parallelized using OpenMP. OpenMP helps to add concurrency to the programs written even in C or Fortran. With the parallel program is possible to try more than one key at the same time (the number of keys depends on the quantity of cores in the processor).

After trying the algorithm, it was possible to realize that the speed up in the parallel program compared with the sequential is 3.7 on average.

Even though the decreasing in the execution time, at attacking with four processors the processing time is really high (more than a hundred thousand years) at trying the keys universe, that’s why we can´t attack DES with just four processors.

Dedicatorias

A mis padres, por siempre estar a mi lado y apoyar mis decisiones.

A mis hermanos, que son y siempre serán parte importante de mi vida.

A mis amigos, que son los hermanos que yo elegí.

Glosario

Computadora paralela: Es un sistema de cómputo multi-procesador que soporta programación paralela.

Criptografía (del griego *krypto*, oculto, y *graphos*, escribir, literalmente “escritura oculta”) es la técnica, bien sea aplicada al arte o la ciencia, que altera las representaciones lingüísticas de un mensaje.

Llave. Es una pieza de información que controla la operación de un algoritmo de criptografía. Esta información es una secuencia de números o letras mediante la cual, en criptografía, se especifica la transformación del texto plano en texto cifrado, o viceversa.

Data Encryption Standard (**DES**) es un algoritmo de cifrado, es decir, un método para cifrar información, escogido por FIPS en los Estados Unidos en1976, y cuyo uso se ha propagado ampliamente por todo el mundo.

FPGA. (Del inglés *Field Programmable Gate Array*) es un dispositivo semiconductor que contiene bloques de lógica cuya interconexión y funcionalidad puede ser configurada “in-situ” mediante un lenguaje de descripción especializado.

Modelo fork-join. Paradigma que proviene de los sistemas Unix, donde una tarea muy pesada se divide en K hilos (fork) con menor peso, para luego "recolectar" sus resultados al final y unirlos en un solo resultado (join).

OpenMP es una interfaz de programación de aplicaciones (API) para la programación multiproceso de memoria compartida en múltiples plataformas. Permite añadir concurrencia a los programas escritos en C, C++ y Fortran sobre la base del modelo de ejecución fork-join.

Programación paralela: Es la programación en un lenguaje que permite explícitamente indicar como porciones diferentes del programa pueden ser ejecutados concurrentemente por diferentes procesadores.

Thread: *Hilo de ejecución.* También llamado *subproceso*. Es una característica que permite a una aplicación realizar varias tareas a la vez (concurrentemente). Los distintos hilos de ejecución comparten una serie de recursos tales como el espacio de memoria, los archivos abiertos, situación de autenticación, etc.

Texto Cifrado: Es aquel texto que ha sido cifrado con algún algoritmo determinado y clave de cifrado.

Texto Plano. Es un archivo informático compuesto únicamente por texto sin formato, sólo caracteres. Estos caracteres se pueden codificar de distintos modos dependiendo de la lengua usada.

Índice general

Índice de figuras y tablas

Figuras

Tablas

CAPÍTULO 1
INTRODUCCIÓN

1.1 Historia de la Criptografía

La información es uno de los bienes intangibles más valiosos de una organización, y desde prácticamente el invento de la escritura, formas de protegerla han sido buscadas; debido a que surgió la necesidad que parte de su información sea sólo conocida por las personas a quienes esta destinada. La necesidad de poder enviar mensajes de modo que únicamente fueran entendidos por los destinatarios, hizo que cada civilización, utilizara códigos secretos y métodos para ocultar la información que se consideraba valiosa, de modo que no fuera descifrable por aquellas personas que puedan interceptar estos mensajes. [1]

En Egipto, la criptografía se encuentra en jeroglíficos no estándares tallados en monumentos. Sin embargo, no se piensa que sean intentos serios de comunicación secreta. En la Grecia clásica, los espartanos transmitían sus mensajes utilizando el *escitalo*, método que consistía en enrollar en un bastón una cinta de papel y sobre la cual se escribía el mensaje a ser transmitido, posteriormente se desenrollaba y así se mandaba a su destino, lugar en el que se enrollaba con un bastón idéntico al primero para así poder entender el mensaje original [2]. Durante el imperio Romano, Julio César empleó un código secreto que consistía en sustituir cada letra del mensaje por otra que en el alfabeto estuviese desplazada tres lugares, este método es conocido como "*cifrado de César*". Siglos más tarde, con base a la idea de este cifrado, se generaron otros como el "*Cifrado de Vigenère*" en 1586, "*Cifrado de Beaufort*" en 1710 y el "*cifrado de Vernam*" en 1917 que empleaba un alfabeto binario [3].

En el siglo XVI Leon Battista Alberti presentó un manuscrito en el que describe un disco cifrador con el que es posible cifrar textos sin que exista una correspondencia única entre el alfabeto del mensaje y el alfabeto de cifrado, con este sistema, cada letra del texto claro podía ser cifrada con un carácter distinto dependiendo de una clave secreta. En el siglo XIX se comienzan a desarrollar sistemas de cifrado con las características poli alfabéticas propuestas por Alberti, que consiste en tener mensajes cifrados con alfabetos que pueden ser diferentes al usado por el mensaje original e incluso de tamaños diferentes; entre los que destacan el de discos concéntricos de *Wheatstone* en 1860 y el de cilindros de *Bazeries* en 1891. [2]

Durante las dos guerras mundiales, la criptografía adquirió un alto grado de perfección. Destaca el uso de la máquina *Enigma*, la cual era usada por el partido nazi para cifrar la mayor parte de sus comunicaciones secretas [4], consistía de un banco de rotores montados sobre un eje, en cuyos perímetros había 26 contactos eléctricos, uno por cada letra del alfabeto inglés. También destacó la máquina *Hagelin,* en las que a través de ruedas con piñones, realiza una cifra similar a la utilizada por el sistema de Beaufort [3]

Con el surgimiento de las computadoras y redes de comunicación la protección de la información se volvió una tarea más complicada, organizaciones gubernamentales se dieron a la tarea de buscar nuevos y novedosos algoritmos para ello.

Hasta la Segunda Guerra Mundial, todos los cifrados eran de clave secreta o simétricos, a finales de los años 40, el pionero en la Teoría de la Información Shannon, sugirió el uso de una mezcla de transposiciones y sustituciones, lo cual se conoce como cifrado de productos [5]. En los años setentas, IBM retomó la sugerencia de Shannon y desarrollo un sistema al que bautizó como *Lucifer*. En 1973 el NBS (Nacional Bureau of Standards) organizó un concurso solicitando un algoritmo de encriptación de datos, teniendo como respuesta de IBM el sistema

antes mencionado. En 1974, después de analizar las propuestas y determinar que ninguna parecía adecuada, realizó otra petición. IBM presentó el criptosistema de Estándar de Encriptación de Datos mejor conocido como DES (por sus siglas en Inglés, Data Encryption Standard), que después de ciertas modificaciones por parte de la NSA (National Security Agency), fue adoptado y publicado por la Federal Information Processing Standard en 1977 (se puede encontrar en el *FIPS 46-3*).

DES fue el primer estándar internacional de cifrado y la publicación de sus especificaciones, por la NBS, estimuló una explosión del interés público y académico por la criptografía. Es de interés mencionar que desde su nacimiento DES fue criticado; al punto de que no se consideraba que fuese a ser utilizado por un largo periodo, sin embargo, fue durante varios años saliendo de la norma hasta julio de 1998. En la actualidad, y aunque DES ya no es la norma, sigue siendo utilizado en algunos medios públicos y privados, como por ejemplo, en el área legal [6].

Debido a su popularidad entre las oficinas gubernamentales de los Estados Unidos y al relativamente pequeño tamaño de llave, DES se convirtió en uno de los objetivos principales de los ataques. [7]

Se han intentado diferentes tipos de ataques, desde aquellos basados en segmentos de texto plano y cifrado conocidos hasta los llamados ataques de fuerza bruta, siendo este el más práctico a la fecha dado que prueba cada una de las posibles llaves 2 razón de 56 [8].

En 1997 los laboratorios RSA convocaron un concurso que consistía en un ataque por fuerza bruta a DES [9]. Tuvo éxito y los participantes utilizaron máquinas de propósito específico y/o sistemas de cómputo distribuido para lograrlo.

Los ataques a DES han existido desde la aparición del protocolo, debido a su importancia como algoritmo de cifrado para las agencias de gobierno. La más reciente descripción detallada se encuentra en el documento escrito por Michael Wiener de Bell Northern Research en 1993. El documento de Wiener incluía un diseño de hardware detallado de una máquina rompe códigos DES (DES Cracker) construida con chips construidos por el usuario. Un diseño completo habría costado cerca de un millón de dólares y pudo haber determinado una llave DES de un texto plano y texto cifrado conocidos en un promedio de 3 horas y media (7 horas en el peor de los casos). La última actualización de este documento es de 1998 y se puede consultar en [10].

Otro método para atacar a DES fue el diseñado por Ian Goldberg y David Wagner de la Universidad de California en Berkeley. Su diseño usó un dispositivo FPGA, es decir, un chip re-programable después de ser manufacturado en una gran variedad de circuitos. A diferencia con la máquina de Wiener, la de Goldberg es más barata debido a la posibilidad de utilizar en su fabricación chips de propósito general. Su diseño se puede consultar en [10].

1.2 Planteamiento del problema

El propósito de la presente investigación es realizar un ataque de fuerza bruta al algoritmo DES, que a diferencia de otros ataques se realizará con software, paralelizando el programa que será ejecutado en una computadora con múltiples procesadores, lo cual permitirá probar varias llaves simultáneamente reduciendo así el tiempo de procesamiento.

Cuando se habla de paralelizar un programa, significa que se modificará el código de tal manera que se indicaran a diferentes procesadores que se encarguen de segmentos del procesamiento.

El lenguaje empleado para la paralelización es OpenMP.

OpenMP es un lenguaje de programación que trabaja a nivel procesador utilizando todos los posibles núcleos (procesadores) con que cuenta que permite realizar la paralelización de un programa. Posteriormente se muestran resultados del tiempo de búsqueda de la llave utilizando el programa secuencial y una comparación con la programación paralela.

1.3 Objetivo General

Realizar un algoritmo paralelo que permita realizar un ataque de fuerza bruta al algoritmo de cifrado DES y romperlo.

1.3.1 Objetivos Particulares

- Diseñar un algoritmo secuencial que permita descifrar la información de una llave a la vez.
- Diseñar un algoritmo paralelo basado a partir del algoritmo secuencial secuencial que permita descifrar múltiples llaves a la vez.
- Evaluar los tiempos de ejecución del programa secuencial y el paralelo bajo las mismas condiciones.
- Comparar los resultados de las evaluaciones de los programas para determinar la eficiencia de cada uno.

1.4 Justificación

Durante muchos años DES fue el algoritmo de cifrado elegido por las agencias de gobierno de Estados Unidos, razón por la cual, la confianza en este creció y se utilizó en empresas de la iniciativa privada, es debido a esto que en la actualidad es posible encontrar sistemas de información que utilicen a DES como sistema de cifrado y por ende, muchos lenguajes de programación cuentan con bibliotecas de funciones con las cuales puede ser implementado fácilmente.

Para realizar ataques de fuerza bruta a sistemas que utilicen al algoritmo DES para el cifrado, ha sido necesario construir maquinas de propósito específico y / o utilizar sistemas de computo distribuido. Ambas opciones requieren de una inversión monetaria alta pero se asegura que el tiempo requerido para romper el algoritmo sea relativamente corto.

Otra opción para realizar ataques de este tipo, es utilizando la programación paralela; opción que hasta la fecha ha sido poco explorada; debido a que hasta hace pocos años las computadoras con las que se contaba a nivel doméstico tenían un solo procesador. Actualmente y con las máquinas multiprocesadores que han salido al mercado, la paralelización del ataque se convierte en una opción viable que vale la pena explorar.

1.5 Propuesta de Solución

Para realizar el ataque de fuerza bruta al algoritmo DES por medio de la programación, es necesario crear un programa que consuma poco tiempo máquina probando cada llave. Para que esto sea posible se debe analizar cada uno de los pasos del algoritmo original con la finalidad de eliminar los pasos que no aumentan seguridad pero sí el tiempo de ejecución. Después de contar con el programa que cumpla con las especificaciones se busca un lenguaje de

programación (de entre los existentes en el mercado) adecuado para paralelizar el programa secuencial, logrando con esto aprovechar las ventajas de las computadoras con múltiples procesadores. Una vez paralelizado el programa es necesario realizar pruebas de rendimiento para determinar que tan viable es la propuesta de solución paralelizada en cuestión tiempo.

1.6 Organización de la tesis

Esta tesis se encuentra compuesta por 6 capítulos organizados de la siguiente manera:

1. El primer capítulo proporciona una introducción del trabajo a desarrollar y resaltando la importancia del mismo. Se presentan los objetivos de este, la justificación del proyecto de tesis y el planteamiento del problema.

2. El segundo capítulo contiene los conceptos básicos de criptografía, la descripción detallada del algoritmo DES, así como los conceptos necesarios de la programación paralela y el lenguaje de programación OpenMP. Finalmente se incluye información sobre algunos antecedentes de ataques realizados al algoritmo DES.

3. En el tercer capítulo se expone la forma en que el programa fue desarrollado. Se mencionan los pasos que se omiten, las razones para hacerlo. Además se presentan el algoritmo secuencial y su versión paralelizada.

4. El cuarto capítulo, muestra las pruebas realizadas al programa desarrollado y los resultados de las mismas.

5. El quinto capítulo contiene las conclusiones del trabajo enfatizando las aportaciones de esta tesis y los trabajos futuros que pueden desarrollarse tomando como base la misma.

CAPÍTULO 2
MARCO TEÓRICO

En este capítulo se darán a conocer los conceptos básicos de criptografía, para posteriormente presentar una descripción detallada del sistema de cifrado DES, debido a que es el algoritmo que se atacará durante el desarrollo del presente trabajo. También se presentan algunos conceptos básicos de cómputo paralelo que nos permitirán comprender la razón de utilizar la paralelización en el presente trabajo. Por otro lado, se podrá encontrar una descripción de las características de OpenMP, que es el lenguaje utilizado para paralelizar el programa a utilizar para realizar el ataque. Además se mencionan algunos de los principales ataques que se han realizado en contra de DES.

2.1 Criptografía

Criptografía es un término que define el estudio de métodos de envío de mensajes enmascarados, es decir, mensajes que deben pasar por un proceso de cifrado, y que solamente los destinatarios tendrán la clave y el método para retirar la mascara y leer el mensaje original. [11]

El objetivo principal de la criptografía es permitir a los usuarios comunicarse sobre un canal inseguro de tal forma que un posible escucha no pueda entender lo que se está diciendo.

El criptoanálisis se refiere a encontrar la llave usada para enmascarar el mensaje. Un criptosistema se usa para cifrar datos. El dato original se conoce como texto plano, y el resultado del cifrado es el texto cifrado. Se descifra el texto cifrado para recuperar el texto plano original. Generalmente en los criptosistemas se requiere de una llave para realizar los procesos de cifrado – descifrado. Lo anterior está representado en la Figura 2.1.

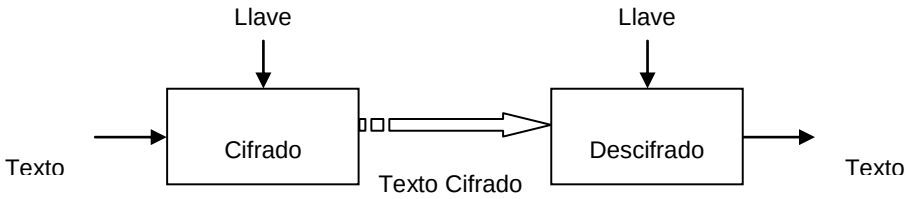


Figura 2.1. Proceso de Cifrado

Existen los cifrados simétricos, donde se utiliza la misma llave para cifrar y descifrar; y los cifrados asimétricos donde llaves diferentes se usan para los procesos de cifrado y descifrado.

Con cualquier cifrado, el objetivo es tener un sistema donde la llave sea necesaria para recobrar el texto plano del texto cifrado; con lo que incluso si el atacante conoce perfectamente el algoritmo usado y otro tipo de información, no podrá recobrar el texto plano sin la llave. [12]

2.2 DES

2.2.1 Historia

El Algoritmo de Cifrado de Datos DES (Estándar de Encripción de Datos) es un algoritmo que cifra bloques de 64 bits, mediante permutación y sustitución, usando una llave de 64 bits, de los cuales 8 son de paridad, es decir, se tienen sólo 56 bits efectivos; produciendo así 64 bits cifrados. DES fue publicado por primera vez en el Registro Federal del 17 Marzo de 1975 (mediante el FIPS 46-2).

Posteriormente, tras múltiples discusiones públicas DES fue adoptado como un estándar para aplicaciones sin clasificar el 15 de Enero de 1977. Inicialmente, se esperaba usar DES como estándar por un periodo de entre 10 ó 15 años; sin embargo, debido a sus características permaneció durante más tiempo. Tras su

adopción, fue revisado aproximadamente cada 5 años. Su última renovación fue en Enero de 1999, revisión que fue publicada en el FIPS 46-3 donde además se incluye al protocolo Triple DES [13].

2.2.2 Descripción de Des

El FIPS 46-2 presenta una descripción general de DES, y el algoritmo donde se definen los pasos matemáticos requeridos para transformar los datos en un texto cifrado y para regresar dicho cifrado a su forma original, es decir, convertir un texto plano en un texto cifrado y viceversa [14].

El algoritmo de DES es simétrico, ya que el descifrado se puede realizar utilizando la misma llave que se uso para el cifrado pero alterando los bits de la llave, de tal forma que el proceso de descifrado sea la inversa del de cifrado. En la *figura 2.2* se presenta un esquema general del algoritmo DES que se irá describiendo detalladamente a continuación.

Durante la primera etapa del algoritmo, el bloque a cifrar (conocido como texto plano) debe someterse a una permutación inicial IP, la cual está dada por la tabla 2.1.

Tabla 2.1. Permutación Inicial (IP)

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

El proceso de aplicación de las permutaciones es como se indica a continuación. Se toma el bit 58 del texto y se coloca en la primera posición, el bit 50 en la segunda, el bit 42 en la tercera y así sucesivamente. En forma general al aplicar la permutación la cadena resultante tiene la siguiente forma:

IP(Texto Plano) = (c58, c50, c42, c34, c26, c18, c10,..., c23, c15, c7).

Posteriormente pasa por un proceso de cálculo dependiente de la llave que será descrito más a delante. La última etapa es otra permutación, que es conocida como IP^{-1} debido a que es exactamente la inversa de la primera, se puede ver la tabla de esta permutación en la *tabla 2.2*. La aplicación de esta tabla es exactamente igual a la de la *tabla 2.1*. Al aplicar la permutación final la cadena resultante vuelve a tener un tamaño de 64 bits.

Tabla 2.2. Permutación Final (**IP^{-1}**)

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

Durante la penúltima etapa se intercambian los 32 bits de la izquierda y los 32 de la derecha. [7].

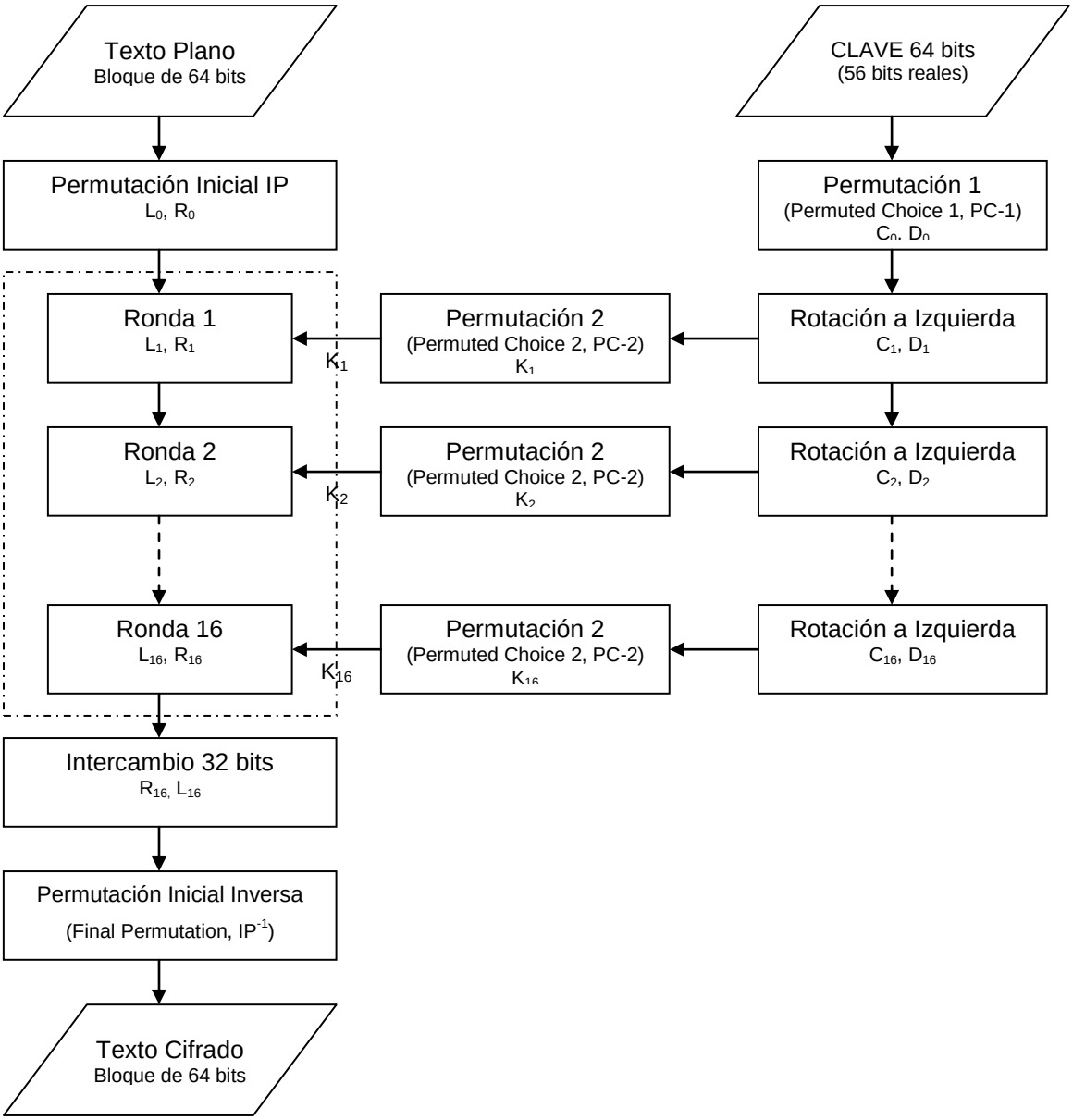


Figura. 2.2. Diagrama General del algoritmo

El cálculo dependiente de la llave consta de 16 rondas, cada una de las cuales cuenta con dos fases. La primera es la que podemos llamar *función de cifrado*, y la segunda se puede llamar *generador de llave*.

Se emplea un valor distinto de subllave (Ki) por ronda, que es obtenido a partir de la llave original de 64 bits, ese proceso se representa en la figura 2.3.

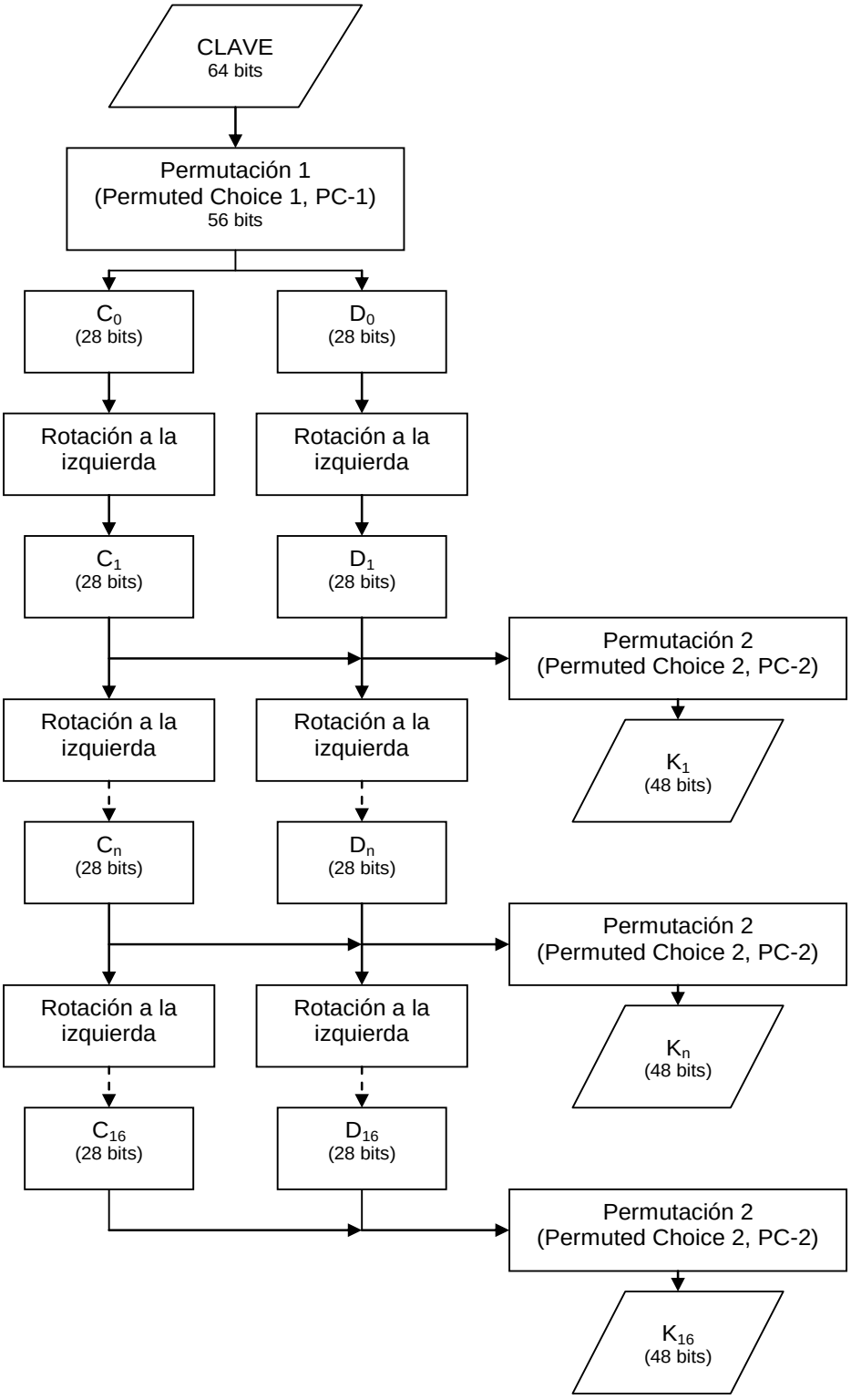


Figura 2.3. Cálculo de las subllaves, K_i

A la llave original en la etapa inicial del proceso de obtención de Ki se aplica una permutación conocida como PC-1, la cual elimina los bits de paridad con lo que se reduce la longitud de la llave de 64 bits a 56.

La tabla de esta permutación se presenta a continuación y se aplica de forma similar a la permutación IP.

Tabla 2.3. Permutación 1 (Permuted Choice 1)

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

Como se puede notar en la tabla anterior, los bits de paridad no aparecen en la permutación, y al no ser tomados en cuenta son eliminados, reduciendo el tamaño de la cadena resultante del proceso de aplicación de la permutación.

Después de aplicada la permutación se debe dividir por la mitad la cadena resultante, con lo que obtendremos a C_0 y D_0 de un tamaño de 28 bits. Cada una de estas cadenas se rota a la izquierda un número de bits que no es siempre el mismo, podemos consultarlo en la tabla 2.4.

Tabla 2.4. Número de rotaciones izquierdas por ronda.

Número de Iteración	Número de Rotaciones Izquierdas
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

De la tabla anterior podemos decir que para pasar de las C_0 y D_0 originales a C_1 y D_1 (de las que obtendremos K_1), se aplicará una rotación, una más para obtener a C_2 y D_2 a partir de C_1 y D_1, pero dos para obtener a C_3 y D_3 a partir de C_2 y D_2.

Ki, es el resultado de la aplicación de la permutación llamada PC-2, que permite la elección permutada de 48 de los 56 bits de las cadenas C_i y D_i (C_{i-1} y D_{i-1} rotadas). En la *tabla 2.5* se muestra la permutación PC-2.

Tabla 2.5. Permutación2 (Permuted Choice 2)

PC-2

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

Para utilizar la tabla anterior, se toman las cadenas C_i y D_i como una sola, que inicia en la posición 1 y termina en la 56. Para formar la cadena resultante de la aplicación de esta permutación, que será la llave buscada (K_i), se toma el elemento 14 de esa cadena y se coloca en la primer posición de la llave, el elemento 17 se coloca en la segunda posición y así sucesivamente hasta que el elemento 32 se coloca en la última posición de la cadena.

Como se puede ver en la *figura 2.3*, para calcular las K_i subsecuentes se deberá aplicar a cada Ci y Di (producto de las rotaciones necesarias) la permutación PC-2 mencionada anteriormente.

Hasta el momento sólo hemos calculado Ki (llave). Ahora se definirá cual es proceso por el que pasa en cada una de las rondas. Este proceso esta plasmado gráficamente en la *figura 2.4*.

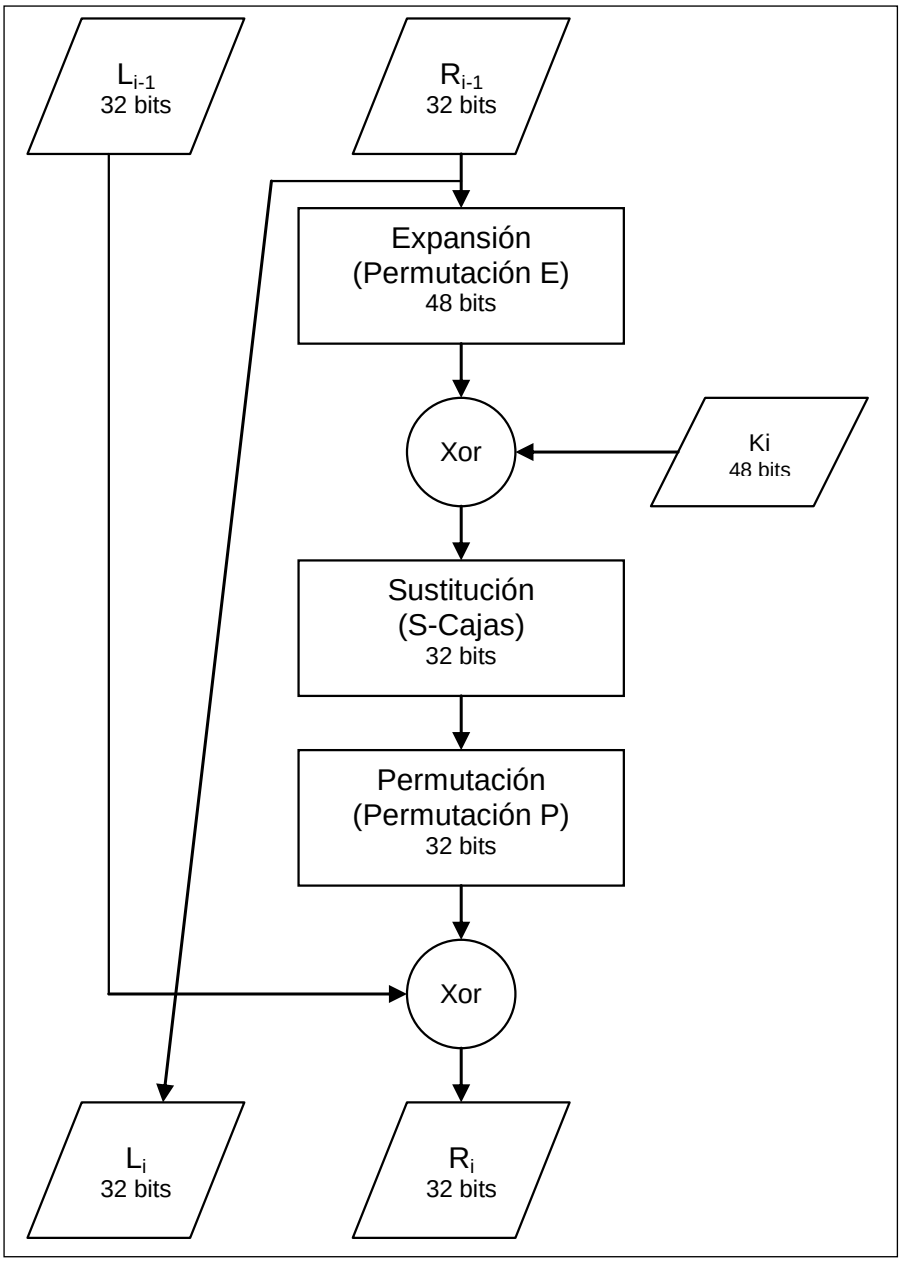


Figura 2.4. Ronda del Algoritmo DES

Para la primer ronda del algoritmo se toma a la cadena resultante de la permutación IP de 64 bits y se divide en dos subcadenas (cada una de 32 bits) a las que se les llama L_{i-1} y R_{i-1}.

El primer paso de cada ronda consiste en tomar a la subcadena R_{i-1} y aplicarle una permutación de expansión conocida como E, que además de intercambiar los bits

aumenta el tamaño del bloque de 32 a 48 bits. La función de expansión se puede ver en la *tabla 2.6*. La aplicación de esta función de expansión E es similar a la de la permutación IP.

Tabla 2.6. Expansión - E

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

A la cadena resultante de la función de expansión y la llave Ki (llave de la ronda), se le aplica una operación X-or. La tabla de verdad de la operación X-or se puede consultar en *la tabla 2.7*.

Tabla 2.7. Tabla de verdad X-or.

A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

Posteriormente, al resultado de la operación X-or se le aplican las cajas S con las que se regresa el tamaño del bloque a 32 bits.

Tabla 2.8. Cajas S

Caja S1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Caja S2

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

Caja S3

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

Caja S4

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

Caja S5

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

Caja S6

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

Caja S7

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

Caja S8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Las cajas S son ocho y se aplican de la siguiente forma. Se divide a la cadena resultante de la operación X-or en 8 bloques de 6 bits cada uno, como se muestra en la *figura 2.5.*

B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8
011011	110001	101001	001100	011011	100111	000011	110011

Figura 2.5. Cadena dividida para la utilización de cajas S.

Se aplicaran las cajas S a los bloques B de manera correspondiente, es decir, para el bloque B_1 se debe utilizar la caja S_1, para el B_2 la caja S_2, y así sucesivamente. En la tabla 2.8 se presentan todas las cajas S.

La aplicación de las cajas S aunque es un poco confusa se puede ilustrar con un ejemplo.

Sean B_1, B_2,…, B_8 las cadenas presentadas en la figura 2.5, y tomando como base para el ejemplo el bloque B_1 y su respectiva caja S_1; se tiene que, para utilizar la caja S_1 se debe dividir el bloque de 6 en dos bloques, uno de 2 bits y otro de 4. Para el bloque de 2 bits se utilizan el primer y último bit, para el de 4 se usan los bits restantes como se muestra en la figura 2.6.

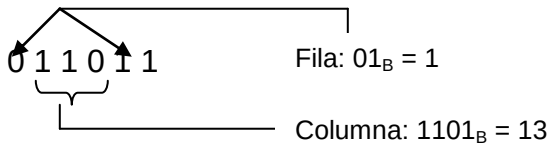


Figura 2.6. División del bloque de 6 bits para utilizar la caja S

Después de divididos los bloques, se toma el de dos bits y se transforma a su valor equivalente en decimal y este representa la fila a utilizar de la tabla (0, 1, 2 ó 3). Con el bloque restante, al convertirlo a decimal, se obtiene el número de la columna a utilizar de la tabla (que va desde 0 hasta 15). En el caso de nuestro ejemplo se obtuvieron la fila 1 y columna 13, la cual tiene un valor de 5 en decimal

que equivale a la cadena 0101 en binario, luego entonces se sustituye a la cadena 011011 por 0101. Haciendo el proceso anterior con todos los bloques de 6 bits al final se tiene una cadena de 32 bits, como se puede apreciar en la figura 2.7.

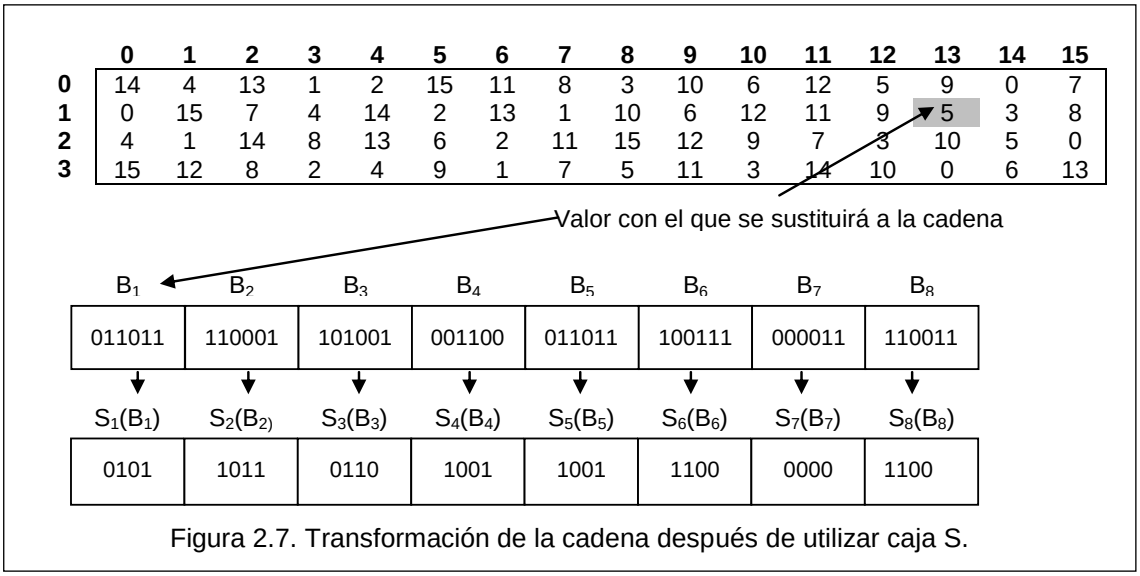


Figura 2.7. Transformación de la cadena después de utilizar caja S.

Al bloque resultante de las cajas S, se le efectúa una nueva permutación (conocida como P). La permutación se aplica según la *tabla 2.9*.

Tabla 2.9. Permutación - P

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

Se utilizará de forma similar a como se han aplicado las otras permutaciones en el algoritmo.

Para finalizar se aplica otra operación X-or al bloque resultante de la última permutación y el bloque L_{i-1} con lo que finalmente obtendremos al bloque Ri. Para la siguiente ronda el bloque Li será el bloque R_{i-1}.

Como se puede notar al final de todo este proceso se tendrán nuevamente los bloques L y R para iniciar la siguiente ronda.

2.3 Programación Paralela

2.3.1 Computo Paralelo

Al hablar de cómputo paralelo nos referimos al uso de una computadora paralela para reducir el tiempo requerido para resolver un problema computacional sencillo. El cómputo paralelo actualmente es considerado por los científicos computacionales e ingenieros en áreas donde cálculos complejos son necesarios como una forma estándar para resolver problemas.

Una computadora paralela esta formada de un sistema de múltiples procesadores que da soporte a programas paralelos. Dos importantes categorías de computadoras paralelas son multi-computadoras y multiprocesadores centralizados. [15]

Multi-computadora, como su nombre lo indica, es una computadora paralela construida a partir de múltiples computadoras y una red de interconexión. Los procesadores en las diferentes computadoras interactúan pasando mensajes unas a otras. Estas se usan en las aplicaciones conocidas como de computo distribuido.

Por otro lado, un multiprocesador centralizado o multiprocesador simétrico (SMP) es un sistema altamente integrado en el cual todos los procesadores comparten

acceso a una memoria global única, la cual soporta comunicación y sincronización entre procesos.

2.3.2 Importancia de las Computadoras Paralelas

Sin importar cuan rápidas sean las computadoras, día a día la tecnología busca formas de hacerlas aún más rápidas. Computadoras con mayor poder de cómputo conducen a nuevos tipos de aplicaciones, las cuales a su vez requieren mayor velocidad.

Para responder a la necesidad del poder de cómputo actual, han surgido computadoras hechas de múltiples componentes que pueden operar simultáneamente y con tareas específicas. Ahora se tiene computadoras capaces de buscar un dato en memoria, multiplicar dos números de coma flotante y evaluar condicionales, todo esto simultáneamente. Aunque a bajo nivel esto se considera procesamiento paralelo y es conocido como *paralelismo a nivel de instrucción*. Los procesadores que lo soportan son aquellos con arquitectura superescalar. [4]

El reordenamiento de las operaciones puede mantener a los componentes de la máquina ocupados. La mayor parte del trabajo de reordenamiento es realizado por el compilador (aunque puede ser apoyado por el hardware). Para poder cumplir con esto los creadores de los compiladores desarrollaron técnicas para determinar las dependencias entre operaciones y para encontrar un ordenamiento que eficientemente utilice el paralelismo a nivel de instrucciones y mantengan muchas unidades funcionales y direcciones a memoria ocupados con trabajo útil. Los compiladores modernos ponen un considerable esfuerzo en la optimización a nivel de instrucción. Existen estudios que muestran que aplicaciones no suelen contener más de tres o cuatro instrucciones diferentes que pueden ser alimentadas a las computadoras de tal forma que se aprovechen al máximo los ciclos de reloj.

En los 80’s se crearon máquinas que consistían de múltiples procesadores completos con una memoria compartida común. Estas máquinas de memoria compartida paralela (o multiprocesador) podían trabajar en diversas tareas a la vez, simplemente distribuyéndolas en los diferentes procesadores. Pueden procesar programas con diferentes necesidades de memoria, y con diferentes cargas de trabajo. Como resultado se volvieron populares en el mercado de los servidores, donde se han mantenido hasta la fecha. Actualmente existen computadoras paralelas de memoria compartida con dos o cuatro procesadores, pero también existen algunas con más de mil en uso y el número está aumentando [16].

En los últimos años los componentes utilizados para construir las computadoras han decrementado en tamaño continuamente, al grado de poder meter 10 millones de transistores en un chip. Por otro lado la velocidad del reloj ha ido en aumento convirtiéndose en un factor importante para incrementar el desempeño del procesador; esto tiene limitaciones inherentes como son la energía consumida y el calor emitido.

Recientemente los diseñadores de computadoras han comenzado a buscar otras estrategias para incrementar el desempeño del hardware, haciendo un mejor uso del espacio disponible en el chip. Dado que añadir unidades funcionales ha sido de poca utilidad, se ha optado por la idea de los procesadores múltiples que comparten memoria, configurados en una sola máquina y en un chip. Esta nueva generación de computadoras paralelas de memoria compartida es económica y de propósito general.

Algunos diseños de computadoras recientes permiten a un único procesador ejecutar cadenas de instrucciones de forma intercalada. Múltiples hilos (thread) simultáneos, instrucciones de múltiples aplicaciones intercaladas en un intento de usar más de los componentes de hardware en cualquier momento.

Por ejemplo, la computadora debe sumar dos valores del set de instrucciones y al mismo tiempo buscar un valor de la memoria necesario para realizar una operación de un diferente conjunto de instrucciones. Otras plataformas recientes replican partes substanciales de la lógica de programación en un chip único comportándose de forma muy similar a las máquinas paralelas de memoria compartida. A este tipo de máquinas se les conoce como multi-núcleo. Plataformas de múltiples hilos simultáneos, máquinas multi-núcleos y computadoras paralelas de memoria compartida proveen un sistema de soporte para la ejecución de múltiples hilos. Se deberán utilizar estas tecnologías combinadas para crear computadoras que puedan ejecutar un gran número de hilos.

En la actualidad algunas computadoras personales y laptops tienen múltiples núcleos o hilos. En poco tiempo, los procesadores tendrán varios núcleos y la posible habilidad de ejecutar múltiples hilos en cada núcleo. Es decir, la tecnología de múltiples hilos se está volviendo muy importante. Es vital que las aplicaciones de software sean capaces de hacer un uso efectivo del paralelismo que se tiene en el hardware. A pesar de los grandes avances en la tecnología de compilación, el programador deberá ayudar, describiendo la concurrencia existente en los códigos de la aplicación. [4]

2.3.3 Procesadores de Múltiples núcleos

La tecnología actual de fabricación de procesadores está llegando a sus límites. Cada vez la miniaturización de los componentes del procesador es más difícil (el límite de construcción del silicio ronda los 15-20nm, donde el silicio empieza a ceder por falta de consistencia, ya se ha llegado a los 65nm), el problema de la generación de calor ha aumentado, produciendo que sea más difícil aumentar la

frecuencia principal del procesador. Todos estos problemas dificultan el aumento de rendimiento de los procesadores.

Los procesadores Pentium 4 Prescott, por ejemplo, no sobrepasan los 3.8 GHz, y necesitan grandes disipadores y ventiladores porque generan mucho calor. Por lo anterior no se podía continuar fabricando procesadores de la misma manera, se estaba llegando a un "estancamiento"; era necesario tomar otro camino, utilizar otra variable que hiciera que el rendimiento del procesador aumentará. Entonces, basándose en el procesamiento en paralelo, se empezaron a construir los procesadores multi-núcleo [17].

2.3.3.1 Definición

Los procesadores de múltiples núcleos son aquellos que contiene dentro de su empaque a varios núcleos o "cerebros". La mayoría de los procesadores son mono-núcleo, o sea tienen un solo cerebro. Mientras un procesador mono-núcleo tiene un solo cerebro para ejecutar procesos, un procesador multi-núcleo puede repartir los procesos entre sus varios cerebros para su posterior ejecución.

Si a varios procesadores deben hacer un solo trabajo, y este es divisible, entonces los procesadores lo harán más rápido. Pero si el trabajo no es divisible, entonces solo uno lo hará.

Por eso las aplicaciones que sacan más provecho de estos procesadores multi-núcleo son aquellas que pueden generar muchos hilos de ejecución (thread) como las aplicaciones de audio/video, cálculo científico, juegos, tratamiento de gráficos en 3D, etc. [18]

Pero de todas maneras siempre hay aplicaciones que no se dividen en hilos de ejecución, que no aprovechan por completo estos procesadores. Pero estos procesadores pueden ejecutar varias de estas aplicaciones al mismo tiempo.

Solo cuando uno ejecute una sola aplicación que no sea paralelizable (no se pueda descomponer en hilos) es cuando no se aprovecha el potencial de procesamiento que tienen estos procesadores.

Actualmente muchos programas son poco paralelizables (excepto en los sectores donde se usan supercomputadoras, sistemas distribuidos y paralelos, etc.), pero se pueden ejecutar muchos de ellos a la vez.

Como historia se puede decir que el primer procesador multi-núcleo en el mercado fue el IBM Power 4 en el año 2000. Una alternativa a los procesadores multi-núcleo son los sistemas multiprocesadores, que consisten en una placa madre que podía soportar desde 2 a más procesadores. El rendimiento es muy bueno, pero también es muy costoso.

2.3.3.2 Descripción

Los procesadores de múltiples núcleos se basaron en los sistemas distribuidos, la computación paralela, y las tecnologías como el Hyperthreading; que mostraban como dividir el trabajo entre varias unidades de ejecución.

Procesamiento en Paralelo es la división de una aplicación en varias partes para que sean ejecutadas a la vez por diferentes unidades de ejecución. Se utiliza en Computación Paralela y la Computación Distribuida.

HyperThreading es una tecnología creada por Intel, para los procesadores Pentium 4 más avanzados. El Hyperthreading hace que el procesador funcione

como si fuera dos procesadores. Esto fue hecho para que tenga la posibilidad de trabajar de forma multihilo (multithread) real, es decir pueda ejecutar muchos hilos simultáneamente.

Un procesador con la tecnología Hyperthreading tiene un 5% más de transistores que el mismo procesador sin esa tecnología.

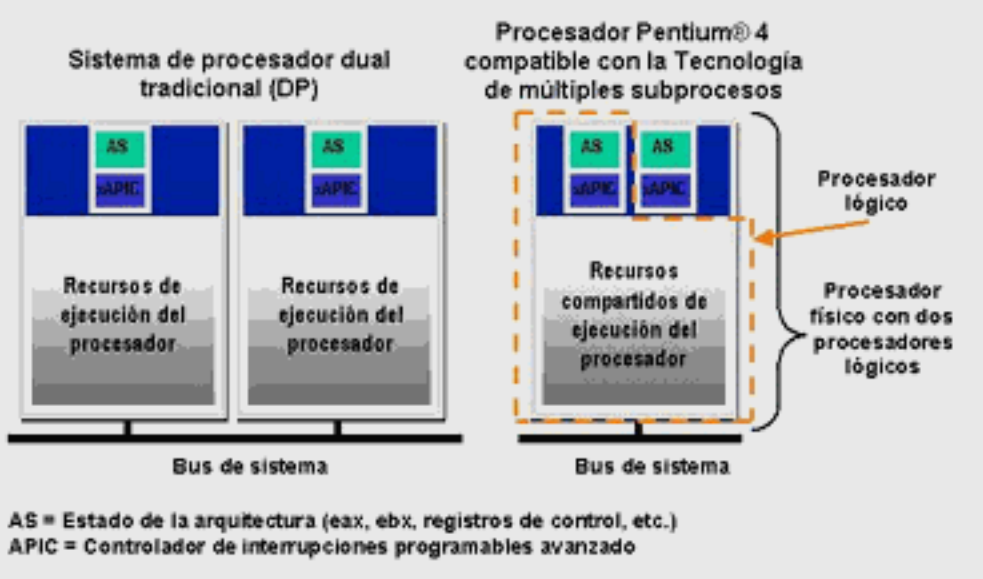


Figura. 2.8. Tecnología Hyperthreading

Los dos procesadores lógicos, que posee el procesador hyperthreading, tienen su propio estado de la arquitectura: registros de control, registros de datos, registros de depuración, etc. y el APIC (controlador avanzado de interrupción programable). Los dos procesadores lógicos comparten la memoria caché, la interfaz del bus del sistema, etc. [18]

2.3.3.3 Procesadores Intel de múltiples núcleos

Intel fabrica procesadores de doble núcleo. Comenzó fabricando los Pentium D en el 2005, luego en el 2006 lanzó los Core Duo y el Core 2 Duo.

Pentium D

Los Pentium D están conformados por dos procesadores Pentium 4 Prescott sin Hyperthreading. Luego Intel sacó el Pentium Extreme Edition (No confundir con el Pentium 4 Extreme Edition) que era un procesador que tenía los procesadores P4 Prescott, con la tecnología Hyperthreading, lo que hacía que el software viera cuatro procesadores.

Las características de los Pentium D son:

- Proceso de fabricación de 90nm
- Tienen la tecnología EM64T, que permite trabajar con 64 bits de forma nativa
- Utilizan núcleos SmithField (basados en los Prescott), cada uno de ellos tiene una memoria caché L2 de 1MB
- Van desde 2.80GHz, del modelo menos potente, hasta 3.20GHz para el modelo más potente.

El procesador que contiene dos núcleos Prescott se llama Smithfield. Los nuevos procesadores de doble núcleo Pentium D se llaman Presler, están construidos con tecnología de fabricación de 65nm y van desde 2.8 hasta 3.73Ghz. Tienen una caché L2 por cada núcleo de 2MB (4MB en total).

Core Duo

Los procesadores Core Duo es una versión para los portátiles, implementa 2MB de caché de memoria compartida para ambos núcleos. Están hechos con la tecnología de 65nm. Su velocidad va desde 1.20 hasta 2.33Ghz. El FSB (bus del sistema) va desde 533Mhz del modelo menos potente hasta 667Mhz para los demás. El gasto de energía va desde 9.0w hasta 31w. Por los datos se ve que tienen una gran relación rendimiento/energía.

Core 2 Duo

Esta nueva familia de procesadores de Intel está basado en la microarquitectura Core, que reemplaza a la antigua microarquitectura Netburst que fue aplicada en los demás procesadores y que ya estaba llegando a sus límites.

La arquitectura Core proviene de la arquitectura que produjo al Pentium M (utilizado por los Intel Centrino), que destaca por el gran rendimiento que obtiene de la poca energía que gasta. El Pentium M además proviene del Pentium III, y este del Pentium Pro (Los Pentium 4 son una rama genealógica aparte).

Las subfamilias del Core 2 Duo son:

- Merom, para portátiles.
- Conroe, para equipos de sobremesa.
- WoodCrest, para servidores.

Los Core 2 Duo salieron en julio del 2006. Además de la versión normal, hay una versión extrema. No se tienen todos los datos disponibles, pero ya se han probado algunos de ellos.

Los Core 2 Duo que han sido probados, por la mayoría de testeadores, son el Core 2 Extreme X6800, el Core 2 Duo E6700 y el E6600. En las pruebas estos procesadores demuestran tener más rendimiento que el más poderoso procesador del AMD, el AMD FX-62. Los más fuertes (X6800 y E6700) vencen en casi todo al FX-62, en algunas pruebas por un margen considerable y el tercero (E6600) está muy cerca. [18]

Actualmente Intel trabaja con una nueva generación de procesadores multi-núcleos, los Core i; entre los que se encuentran los procesadores Intel Core i3, Intel Core i5, Intel Core i7, Intel Core i5 vPro y Intel Core i7 vPro. Entre las principales características de estos procesadores están:

- Automáticamente reducen el consumo de energía cuando no se necesita.

- Tecnología Intel HT**,** que permite que cada núcleo de su procesador trabaje en dos tareas al mismo tiempo.
- Gráficos Intel HD 3000, que proporcionan rendimiento incorporado para videojuegos generales y ocasionales sin necesidad de hardware de gráficos adicional.
- Intel Quick Sync Video, que simplifica la edición y compartición de vídeos.
- InTru 3D, que ofrece visualización de imágenes estereoscópicas en 3D y HD en su portátil en los desplazamientos.
- Funciones de seguridad asistidas por hardware que refuerzan la seguridad del PC.
- Codificación de datos que puede funcionar hasta cuatro veces más rápido gracias a las nuevas instrucciones AES-NI de Intel (nuevas instrucciones del estándar de codificación avanzada).
- Tecnología antirrobo Intel opcional que hace valer la autenticación del software de codificación de datos cuando las computadoras personales salen del estado de reposo, cerrando un conocido hueco de seguridad.
- La tecnología Intel Turbo Boost 2.0 acelera el rendimiento para adaptarse a las cargas de trabajo del usuario.
- Hasta el doble de rendimiento multitarea más rápido al comparar un portátil equipado con un procesador de la 2ª generación de procesadores Intel Core i con una portátil equipada con un procesador Intel Core 2 Duo del 2008.
- Sensacional rendimiento visual sin el coste adicional y los requisitos de consumo de una tarjeta gráfica dedicada [18].

2.3.4 Lenguajes de Programación

Un lenguaje de programación, consiste en un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones.

Un lenguaje de programación permite a uno o más programadores especificar de manera precisa: sobre qué datos una computadora debe operar, cómo deben ser estos almacenados y transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. Todo esto, a través de un lenguaje que intenta estar relativamente próximo al lenguaje humano o natural, tal como sucede con el lenguaje Léxico. Una característica relevante de los lenguajes de programación es precisamente que más de un programador puedan tener un conjunto común de instrucciones que puedan ser comprendidas entre ellos para realizar la construcción del programa de forma colaborativa. Los lenguajes de programación pueden clasificarse empleando distintos métodos y puntos de vista. Esta clasificación se basa en el paradigma que utilizan. [3]

2.3.5 Paradigma de Programación

Un paradigma de programación provee (*y determina*) la visión y métodos de un programador en la construcción de un programa o subprograma. Diferentes paradigmas resultan en diferentes estilos de programación y en diferentes formas de pensar la solución de problemas (con la solución de múltiples “problemas” se construye una aplicación).

2.3.6 Paradigmas de Programación Paralela

Los paradigmas de programación no están unívocamente ligados a las arquitecturas paralelas, aunque sí que guarden relación como se indicará, existen dos paradigmas principales de programar computadores paralelos:

Programación paralela mediante memoria compartida (espacio de direcciones único). El programador escribe el programa, bien empleando directivas de paralelismo de datos bien asumiendo explícitamente diferentes procesos que pueden acceder a un espacio de direcciones único, con variables globales compartidas. Este paradigma es apropiado para arquitecturas de memoria compartida, aunque el espacio de direcciones único también puede ser simulado sobre arquitecturas de memoria distribuida.

Programación paralela mediante paso de mensajes. El programador explícitamente divide el trabajo y los datos entre varios procesos y debe gestionar la comunicación de datos entre ellos.

Esta aproximación es completamente flexible, si bien requiere un mayor trabajo del programador. Puede ser empleado en arquitecturas de memoria distribuida e incluso en redes de computadoras, y también en arquitecturas de memoria compartida, aunque en ese caso los diferentes procesos utilizan partes de memoria independientes y separadas. [16]

2.3.7 Técnicas de Programación Paralela

La programación paralela es programar en un lenguaje que permita explícitamente indicar como diferentes porciones del cómputo deberán ser ejecutadas concurrentemente por diferentes procesadores.

Existen dos tipos de paralelización:

- ***Paralelismo de datos***. Se da cuando procesos independientes aplican la misma operación a diferentes elementos del conjunto de datos.
- ***Paralelismo funcional.*** Se da cuando procesos independientes aplican diferentes operaciones a diferentes elementos del conjunto de datos.

Para programar computadoras paralelas, en forma general existen 4 posibles alternativas:

- Extender un compilador existente para traducir programas secuenciales en paralelos.
- Extender un lenguaje existente con nuevas operaciones que permita a los usuarios expresar paralelismo.
- Agregar una nueva capa de lenguaje paralelo al lenguaje secuencial existente.
- Definir un lenguaje de paralelización y sistema de compilación totalmente nuevo.

Actualmente la opción más popular para paralelizar programas el enfoque de agregar a un lenguaje secuencial existente con bajo nivel de constructores y expresado por llamadas a las funciones o directivas de compilación una nueva capa de lenguaje. La programación paralela en C con OpenMP es un ejemplo de esto. Este enfoque tiene la ventaja de crear programas altamente eficientes y portables en un amplio rango de sistemas paralelos; pero tiene la desventaja de ser más difíciles de codificar y depurar que los escritos en lenguajes paralelos de alto nivel.

2.4 OpenMP

OpenMP permite la creación de programas paralelos de memoria compartida. OpenMP es una interfaz de programación de aplicaciones (API) para la programación multiproceso de memoria compartida multiplataforma. Permite añadir concurrencia a los programas escritos en C, C++ y Fortran. Está disponible en muchas arquitecturas, incluidas las plataformas de Unix y de Microsoft Windows. Se compone de un conjunto de directivas de compilador, rutinas de

biblioteca para programadores de aplicaciones paralelas. Además cuenta con variables de entorno que influencian el comportamiento en tiempo de ejecución.

Es un modelo de programación portable y escalable que proporciona a los programadores una interfaz simple y flexible para el desarrollo de aplicaciones paralelas en diversos tipos de computadoras multiprocesadores.

Entre una de sus más notables ventajas esta que simplifica ampliamente escribir programas multi-hilos en C, C++ y Fortran, además de que estandariza los últimos 20 años de multiproceso simétrico.

2.4.1 Programando Computadoras Paralelas de Memoria Compartida (SMPS)

Una vez que las computadoras paralelas de memoria compartida (o SMPs por sus siglas en ingles -Shared Memory Parallel- Paralelización de Memoria Compartida) pudieron ser fabricadas por un precio moderado, era necesario asegurar que su poder de cómputo pudiera ser explotado por aplicaciones individuales. Los compiladores habían sido hasta el momento siempre los responsables de adaptar un programa para que hiciera mejor uso del paralelismo interno de la máquina. Desafortunadamente, era muy difícil para los compiladores hacerlo para computadoras con múltiples procesadores o núcleos. Se debe a que los compiladores deben identificar independientemente cadenas de instrucciones que pueden ser ejecutadas paralelamente. Existen técnicas para extraer dichas cadenas de instrucciones de un programa secuencial; y para programas simples podría ser útil intentar la opción de paralelización automática de los compiladores. Sin embargo, el compilador frecuentemente no cuenta con suficiente información para decidir si es posible dividir un programa de esta forma. El compilador tampoco podrá hacer grandes cambios en el código, tales como reemplazar un

algoritmo que no sea apto para la paralelización. Es por esto que la mayor parte del tiempo el compilador necesitara ayuda del usuario.

Para tener a los procesadores múltiples colaborando para ejecutar una sola aplicación, las aplicaciones buscan regiones de código cuyas instrucciones pueden ser compartidas entre los procesadores. La mayor parte del tiempo, se enfocan en distribuir el trabajo en ciclos anidados a los procesadores.

En la mayoría de los programas, código ejecutado en un procesador requiere resultados que han sido calculados en otro. Inicialmente, esto no era un problema porque un valor producido por un procesador puede ser almacenado en la memoria principal y recuperado de ahí por el código de otro procesador según sea necesario. Sin embargo, el programador necesita asegurar que el valor sea recuperado después de que ha sido producido, es decir, el acceso es ejecutado en el orden requerido. Dado que los procesadores operan independientemente unos de otros, esta no es una dificultad trivial: sus relojes no están sincronizados, y pueden y ejecutan sus segmentos de código a velocidades ligeramente diferentes [15].

Algunos proveedores de SMPs proporcionaron notaciones especiales para especificar la forma de repartir el trabajo a procesadores individuales de un SMP, así como hacer cumplir un orden de acceso por los diferentes hilos para compartir datos. La notación principalmente toma la forma de instrucciones especiales (o directivas) que pueden ser añadidas a programas escritos en lenguajes secuenciales, especialmente Fortran o C. El compilador usa esta información para crear el código real para su ejecución por cada procesador. A pesar de que esta estrategia funciona, tiene las deficiencias obvias que un programa escrito para una SMP no necesariamente se ejecuta en otro.

Un grupo industrial informal llamado “El Foro del Computo Paralelo” acepto un grupo de directivas que especifican el paralelismo de bucles en programas de Fortran.

OpenMP fue definido por un grupo de vendedores quienes unieron fuerzas durante la segunda mitad de 1990 para proveer medios comunes para programar un gran rango de arquitecturas SMP. La primera versión, consistente de un conjunto de directivas que pueden ser usadas con Fortran, fue liberada al público a finales de 1997.

2.5 Estado del Arte

Desde la aparición del Algoritmo DES, los ataques a este se hicieron presentes. Uno de los primeros ataques a DES fue el conocido como Criptoanálisis Diferencial descrito en [19]. Consiste en elegir un gran número de textos claros, tales que la diferencia entre cada uno de ellos sea fija y conocida. Posteriormente, se estudian las parejas de texto claro-texto cifrado y se analiza cómo esa diferencia se va propagando a lo largo del algoritmo. Con esta técnica, el número de claves posibles se “reduce” a 2^{47} (el número de llaves original era de 2^{56}).

Otro de los ataque más conocidos a DES fue el Criptoanálisis Lineal, propuesto por Mitsuru Matsui en [20]. Este ataque se basa en suponer que el texto cifrado es una función lineal del texto claro y de la clave. Difícilmente algún criptoanalista podría admitir lo anterior, sin embargo, Matsui fue capaz de encontrar funciones lineales que con altas probabilidades satisfacían las sucesivas vueltas de DES, llegando a la conclusión de que se deben probar 2^{43} claves en promedio para hallar la verdadera. En 1994, una red de doce estaciones de trabajo fue capaz, usando el análisis lineal, de descifrar un mensaje cifrado con DES trabajando durante cincuenta días.

Otro ataque que aunque sencillo es muy eficaz, es el conocido como “de fuerza bruta”. Este ataque consiste en probar todas las posibles llaves (2^{56}); lo cual como es de suponerse en una máquina regular con un único procesador se convierte en una tarea si bien no imposible si poco viable debido al tiempo requerido para obtener resultados. [21]

En Enero de 1997, los Laboratorios RSA lanzarón una serie de retos criptográficos cuyas especificaciones se pueden consultar en [9]. El objetivo era encontrar un mensaje secreto, el cual fue encriptado utilizando DES. En total se lanzaron tres retos de este tipo. En el último de ellos, llamado “DES Challenge III” (Reto DES III), la Electronic Frontier Foundation (EFF) [22] con su computadora de uso específico “Deep Crack” en combinación con distributed.net [23], pudo descifrar el mensaje en 22 horas 15 minutos. Ambas compañías publicaron detalles del trabajo en sus respectivas páginas web.

Además de la máquina diseñada por la EFF, existen otras opciones de computadoras de uso especifico diseñadas y utilizadas para realizar ataques de fuerza bruta a DES, especificaciones de algunas de estas pueden ser consultadas en [10] donde la EFF se dio a la tarea de recopilar la información.

Después de estos ataques y aunque el interés por el algoritmo DES decremento, se siguieron realizando algunos otros ataques, debido principalmente a su importancia y amplio uso en empresas donde la información codificada era útil solo a corto plazo y de poco valor económico.

En [10] Michael J. Wiener menciona que después del éxito con los ataques por fuerza bruta a DES en 1997, quedo demostrada la vulnerabilidad de DES. De acuerdo con Wiener el diseño de DES es lento en software pero se vuelve compacto y rápido al ser implementado en hardware. Pero la opción de hardware puede llegar a ser muy cara (alrededor de un millón de dólares).

A pesar de lo dicho por Wiener nuevos ataques a DES por software han surgido. En el 2010, Cao y Liu en [24] presenta un ataque basado en una relación $L_{i+1}= R_i$. El ataque requiere alrededor de 2^{16} textos cifrados del mismo R16, cifrado por la misma llave. La complejidad del ataque (obtenida desde un punto de vista teórico) es de 255. Cao y Liu sostienen que este ataque desde un punto de vista práctico es más viable que el Diferencial o el Lineal.

En [22] se expone un ataque a DES por fuerza bruta basado en GPUs programados con CUDA, el cual explota las ventajas de la arquitectura gráfica multi-núcleos de Nvidia GT200. Se prueba como un GPU supera en un factor de 10 a un CPU. Además se proveen datos donde se muestra que el costo de esta implementación es moderado (en comparación con las máquinas de uso específico) siendo innecesario ser un experto técnico en la materia se pueden obtener resultados en un tiempo relativamente corto. Lo que hace posible ahora a usuarios promedio (sin conocimientos técnicos específicos), realizar ataques por fuerza bruta al protocolo con solo contar con computadoras personales utilizadas típicamente para video juegos.

CAPÍTULO 3

DESARROLLO

3.1 Introducción

En el presente trabajo, se efectuó un ataque de fuerza bruta al algoritmo DES, utilizando un programa paralelizado que deberá probar 2^{56} llaves. Debido a la necesidad de probar una enorme cantidad de llaves en el menor tiempo posible, se realizaron algunos cambios al algoritmo original, dichos cambios se pueden apreciar en la siguiente figura.

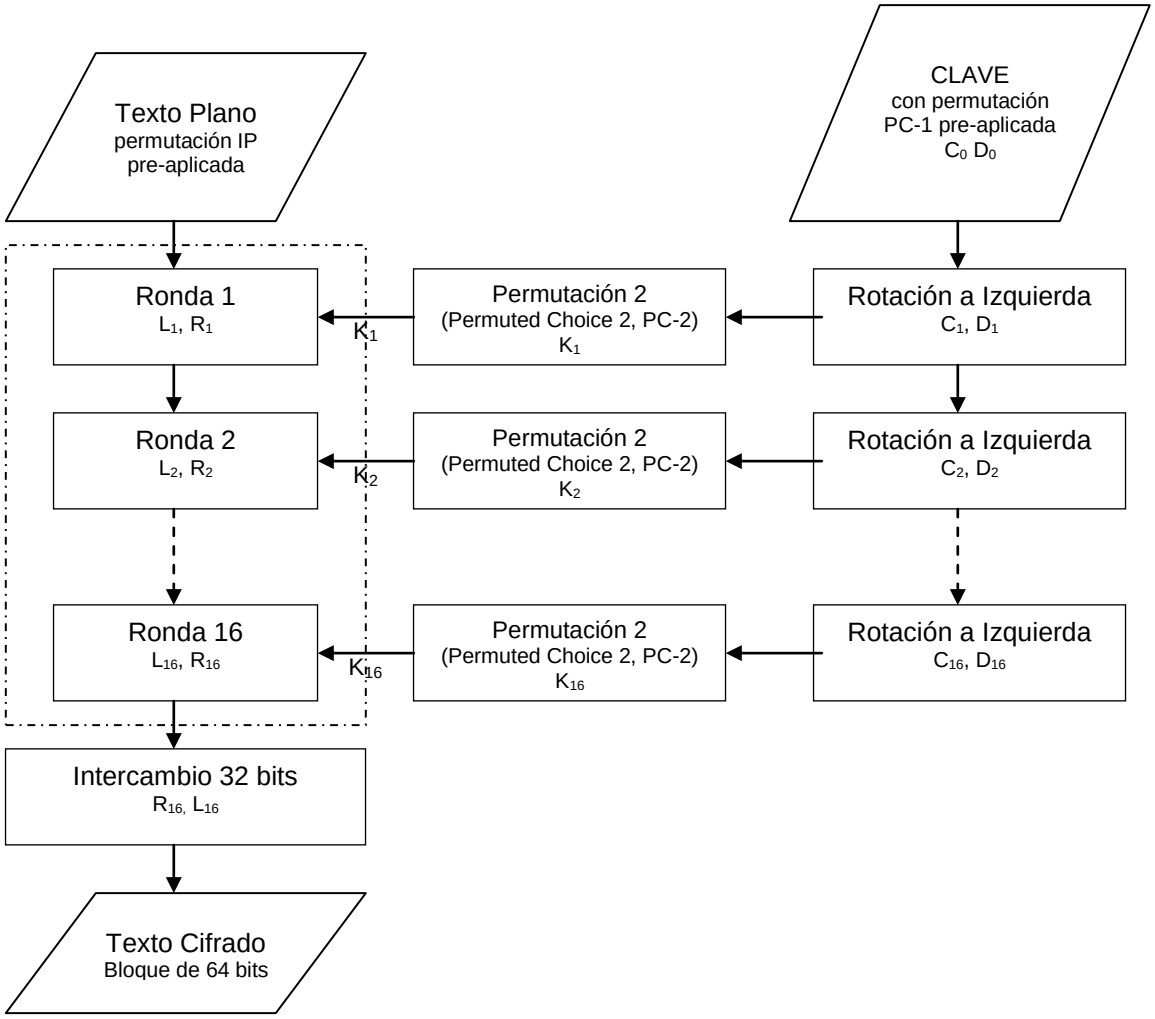


Figura 3.1. Diagrama General del algoritmo DES modificado

Lo primero que se omite son las permutaciones inicial (IP) y final (IP^{-1}) dado que estas no agregan mayor grado de seguridad (complejidad) al algoritmo pero si consumen tiempo de ejecución. Por otro lado en el proceso de generación de las llaves, no se aplica la primera permutación (PC-1) y dado que el universo de llaves efectivo es 2^{56}, la llave con la que se realiza la prueba consta solo de 56 bits (al ser eliminados los bits de paridad considerado la pre-aplicación de la permutación PC-1); la razón para eliminar esta permutación y tomar esa longitud de llave es básicamente la misma que en el caso de las permutaciones inicial y final, no agregan mayor complejidad al algoritmo pero al ser aplicadas consumen tiempo de procesador que será necesario para lograr el objetivo que se persigue.

3.2 Algoritmo del Programa

3.2.1 Pseudocódigo del Algoritmo Secuencial de DES

Para el desarrollo del programa fue necesario, antes que nada, realizar el programa secuencial que aplicara el algoritmo de cifrado DES a un texto claro predeterminado, con una llave cualquiera, para posteriormente ser comparado el resultado de este con el texto cifrado a obtener.

El programa antes mencionado esta basado en el siguiente pseudocódigo. Con esta versión del algoritmo se prueba todo el universo de llaves en un solo proceso.

1. Inicio.
2. Inicializar variable TxtP y TxtC con los textos plano y cifrado, respectivamente, dados por default. t1 + t2
3. Se inicializan Li y Ri (variables para iniciar el proceso de cifrado).
4. Para posibles_llaves=0 hasta posibles_llaves=100000
    a. Se genera la llave a probar (llave).
    b. Para ronda=1 hasta ronda=16
        i. Se genera la llave para la ronda. (llaveR).

ii. Se aplica permutación E a Ri. (E=E(Ri)).
iii. Se aplica una operación OR-EXCLUSIVA al resultado de la permutación E y la llave de la ronda. (E Xor llaveR).
iv. Se aplican las cajas (S-box) (S=S(E Xor llaveR)).
v. Se aplica permutación P (P=P(S))
vi. Se aplica una operación OR-EXCLUSIVA al resultado de la permutación P y a Li. (P Xor Li)
vii. Li+1=Ri
viii. Ri+1= P Xor Li

c. Fin_para
d. resultado_rondas= Ri + Li
e. Si resultado_rondas es igual TxtC
    i. Escribe “Se encontró la llave correcta, que es posibles_llaves”
    ii. Se rompe para de posibles_llaves
f. Si no
    i. Escribe “No se encontró la llave correcta.”

5. Fin_para
6. Fin

3.2.2 Pseudocódigo del Algoritmo Paralelo de DES

Como se pudo apreciar en el pseudocódigo anterior, el trabajo de búsqueda de la llave correcta recae sobre un solo procesador. Ahora se presenta la versión
Con esta versión del algoritmo se balancean la cantidad de llaves a ser probadas por cada proceso.

1. Inicio.
2. Inicializar variable TxtP y TxtC con los textos plano y cifrado, respectivamente, dados por default. t1 + t2
3. Se inicializan Li y Ri (variables para iniciar el proceso de cifrado).
4. **Se indica el número total de llaves *universo=100000.***

5. **Se determina el número de procesos *num_procesadores*.**
6. **Se determina el número de proceso *num_procs*.**
7. **rango = universo / num_procesadores**
   //Por procesador
8. Para **posibles_llaves = num_procs*rango** hasta **posibles_llaves=(num_procs+1)*rango-1**
   a. Se genera la llave a probar (llave). //El valor que alimenta a la función generadora es *posibles_llaves*
   b. Para ronda=1 hasta ronda=16
      i. Se genera la llave para la ronda. (llaveR)
      ii. Se aplica permutación E a Ri. (E=E(Ri))
      iii. Se aplica una operación OR-EXCLUSIVA al resultado de la permutación E y la llave de la ronda. (E Xor llaveR)
      iv. Se aplican las cajas (S-box) (S=S(E Xor llaveR)).
      v. Se aplica permutación P (P=P(S))
      vi. Se aplica una operación OR-EXCLUSIVA al resultado de la permutación P y a Li. (P Xor Li)
      vii. $L_{i+1}=R_i$
      viii. R_{i+1}= P Xor Li

      Fin_para
   c. resultado_rondas= Ri + Li
   d. Si resultado_rondas es igual TxtC
      Escribe “Se encontró la llave correcta, **posibles_llaves**”
      Se rompe para de posibles_llaves

      Si no
      Escribe “No se encontró la llave correcta.”
9. Fin_para
10. Fin

3.3 Especificaciones del Programa

Como se menciono se puede ver en la figura 3.1, se hicieron modificaciones al algoritmo original, pero además y gracias a la forma en que fue programado (utilizando operaciones a nivel de bits) se pudieron realizar otros cambios como los especificados a continuación.

Una de las modificaciones consistió en cambiar la representación de las cajas S. Originalmente el algoritmo marca que para la aplicación de las cajas S es necesario dividir la cadena en paquetes de 6 bits, de los cuales el primero (bit en la posición 0) y el último (bit en la posición 5) sirven para representar filas y los bits centrales para representar filas de las cajas mencionadas. (Para más información sobre la aplicación de las cajas S, ver ejemplo en capitulo II).

Caja S1 Original

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
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
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

Si esta caja se representa con bits nos queda.

Caja S1 Binaria

000000	1110	010000	0011	100000	0100	110000	1111
000001	0000	010001	1010	100001	1111	110001	0101
000010	0100	010010	1010	100010	0001	110010	1100
000011	1111	010011	0110	100011	1100	110011	1011
000100	1101	010100	0110	100100	1110	110100	1001
000101	0111	010101	1100	100101	1000	110101	0011
000110	0001	010110	1100	100110	1000	110110	0111
000111	0100	010111	1011	100111	0010	110111	1110
001000	0010	011000	0101	101000	1101	111000	0011
001001	1110	011001	1001	101001	0100	111001	1010
001010	1111	011010	1001	101010	0110	111010	1010
001011	0010	011011	0101	101011	1001	111011	0000
001100	1011	011100	0000	101100	0010	111100	0101
001101	1101	011101	0011	101101	0001	111101	0110
001110	1000	011110	0111	101110	1011	111110	0000
001111	0001	011111	1000	101111	0111	111111	1101

Al reacomodar los valores ya no por filas y columnas si no por posiciones de acuerdo a los valores decimales en un vector la caja S1 quedara con la siguiente representación.

Caja S1 por posiciones

14	0	4	15	13	7	1	4	2	14	15	2	11	13	8	1
3	10	10	6	6	12	12	11	5	9	9	5	0	3	7	8
4	15	1	12	14	8	8	2	13	4	6	9	12	1	11	7
15	5	12	11	9	3	7	14	3	10	10	0	5	6	0	13

Esta es la representación utilizada en el problema, por lo que en lugar de tomar la cadena de 6 bits, separarla para identificar la fila y la columna correspondiente y buscar el valor equivalente, solo se debe tomar el valor de la cadena de 6 bits leerlo como decimal y buscar la ubicación en el vector.

Aplicando este proceso a todas las cajas, las cajas S usadas en el programa quedaron de la siguiente forma:

Caja S2 por posiciones

15	3	1	13	8	4	14	7	6	15	11	2	3	8	4	14
9	12	7	0	2	1	13	10	12	6	0	9	5	11	10	5
0	13	14	8	7	10	11	1	10	3	4	15	13	4	1	2
5	11	8	6	12	7	6	12	9	0	3	5	2	14	15	9

Caja S3 por posiciones

10	13	0	7	9	0	14	9	6	3	3	4	15	6	5	10
1	2	13	8	12	5	7	14	11	12	4	11	2	15	8	1
13	1	6	10	4	13	9	0	8	6	15	9	3	8	0	7
11	4	1	15	2	14	12	3	5	11	10	5	14	2	7	12

Caja S4 por posiciones

7	13	13	8	14	11	3	5	0	6	6	15	9	0	10	3
1	4	2	7	8	2	5	12	11	1	12	10	4	14	15	9
10	3	6	15	9	0	0	6	12	10	11	1	7	13	13	8

15	9	1	4	3	5	14	11	5	12	2	7	8	2	4	14

Caja S5 por posiciones

2	14	12	11	4	2	1	12	7	4	10	7	11	13	6	1
8	5	5	0	3	15	15	10	13	3	0	9	14	8	9	6
4	11	2	8	1	12	11	7	10	1	13	14	7	2	8	13
15	6	9	15	12	0	5	9	6	10	3	4	0	5	14	3

Caja S6 por posiciones

12	10	1	15	10	4	15	2	9	7	2	12	6	9	8	5
0	6	13	1	3	13	4	14	14	0	7	11	5	3	11	8
9	4	14	3	15	2	5	12	2	9	8	5	12	15	3	10
7	11	0	14	4	1	10	7	1	6	13	0	11	8	6	13

Caja S7 por posiciones

4	13	11	0	2	11	14	7	15	4	0	9	8	1	13	10
3	14	12	3	9	5	7	12	5	2	10	15	6	8	1	6
1	6	4	11	11	13	13	8	12	1	3	4	7	10	14	7
10	9	15	5	6	0	8	15	0	14	5	2	9	3	2	12

Caja S8 por posiciones

13	1	2	15	8	13	4	8	6	10	15	3	11	7	1	4
10	12	9	5	3	6	14	11	5	0	0	14	12	9	7	2
7	2	11	1	4	14	1	7	9	4	12	10	14	8	2	13
0	15	6	12	10	9	13	0	15	3	3	5	5	6	8	11

Otro factor que fue fundamental para reducir tiempos de ejecución en el programa, fue el hecho de que las cadenas empleadas y operaciones sobre estas se realizan a nivel de bits y no con vectores tipo carácter o entero como en una primera versión del programa secuencial. Con lo anterior no sólo se redujo el tiempo de ejecución si no también el número de líneas de código, haciéndolo más entendible.

Una vez terminado y probado el código del programa secuencial, se procedió a paralelizar dicho código, para lo cual se utilizó OpenMP que al trabajar con memoria compartida (y no distribuida como en el caso de MPI) mejora la velocidad de ejecución.

Se debe tomar en cuenta que al probar una llave durante las 16 rondas, los datos de la primera ronda sirven de base para la segunda, los de la segunda se necesitan en la tercera, etc., es decir, los datos son dependientes, por lo que no es posible paralelizar a nivel de funciones. Además si se considera que el mismo proceso de cifrado deberá ser aplicado y probado con un gran número de llaves, esto nos lleva a la conclusión de que el tipo de paralelización a utilizar es a nivel de datos, donde a la vez se probaran en diferentes procesadores diferentes llaves para con esto reducir el tiempo de búsqueda de la llave.

3.4 Complejidad del Algoritmo

Dentro del cómputo paralelo la pregunta fundamental es si cada problema que tiene una solución secuencial viable tiene una solución en paralelo más rápida. Es decir, ¿el paralelismo puede ser siempre usado para acelerar el tiempo de proceso de un problema? Muchos problemas pueden ser resueltos con grandes mejoras en la velocidad cuando son resueltos en paralelo [1].

La complejidad de un algoritmo es una medida de la cantidad de recursos (tiempo, memoria) que el algoritmo necesita. Se expresa en función del tamaño del problema. [4]

En el caso de nuestro algoritmo, se calcula la complejidad temporal, ya que es el tiempo el factor determinante que permitirá que nuestro algoritmo sea o no eficiente.

Se dice que un algoritmo es eficiente cuando logra llegar a sus objetivos planteados utilizando la menor cantidad de recursos posibles, es decir, minimizando el uso memoria, de pasos y de esfuerzo humano.

Al realizar un análisis de los pasos en el algoritmo, se puede determinar que es de **orden cuadrático**, ya que, la mayor parte de las operaciones son lineales, pero se tienen dos ciclos anidados. Matemáticamente lo podemos expresar como: $O(n^2)$.

CAPÍTULO 4

PRUEBAS

4.1 Introducción

En este capítulo se presenta el resultado de las pruebas realizadas a los programas secuencial y paralelo que prueban todas las posibles llaves en el algoritmo DES hasta encontrar aquella donde al cifrar un texto claro predeterminado se obtiene el correspondiente texto cifrado que podrá ser comparado con el preexistente. La programación fue realizada en C para la versión secuencial, y fue modificado con ayuda de OpenMP para su versión paralela con lo que se creo un programa que trabaja con hilos.

Una de las pruebas más importantes a realizar sobre un algoritmo paralelo es su velocidad en comparación con la versión secuencial. Para lo cual es necesario probar el algoritmo bajo las mismas condiciones (cantidad de datos, llave inicial, computadora, etc.).

Se realizaron varias pruebas con diferente cantidad de llaves a probar y se podrán apreciar los resultados de las pruebas y su interpretación según algunas de las leyes de paralelización existentes.

4.2 Pruebas Realizadas

Para el presente trabajo, se desarrolló un programa capaz de probar el universo de llaves existentes por secciones, estableciendo desde código la llave con la que se quiere comenzar a trabajar y el número de llaves a probar. El texto claro y su

correspondiente texto cifrado con el que se comprobara el resultado se preestablecen también desde código. En esta sección se mostraran algunas tablas y gráficas donde se podrán apreciar los diferentes tiempos de ejecución con diversas cantidades de datos.

El tiempo requerido para probar todo el universo de llaves con solo 4 procesadores es muy alto (más de cien mil años); debido a lo anterior no se prueba todo el universo de llaves, pero con el número de pruebas realizadas es posible notar la tendencia del tiempo de ejecución

4.2.1 Pruebas a los algoritmos Secuencial y Paralelo

Los algoritmos se ejecutan desde la consola de Linux, aprovechando las características del comando ***Time***. La sintaxis del comando ***Time*** es, ***Time [comando].*** Devuelve el tiempo de ejecución total, el tiempo que el sistema ha dedicado a ese usuario y el tiempo de preparación del programa o comando pasado como argumento.

Por otro lado, en Linux se puede utilizar el operador **>**, con el cual se solicita que envíe la salida del comando a la izquierda del símbolo, hacia el archivo especificado a la derecha del mismo borrando el contenido del archivo. En el caso del comando que se esta utilizando, al ejecutar el programa ***DES_bitS*** su resultado se almacena en el archivo ***tS.txt***.

Por lo anterior, para ejecutar el programa secuencial se utiliza la siguiente instrucción desde consola de Linux:

fabiola@fabiola-desktop:~$ time ./DES_bitS >tS.txt

Al final nos dará la estadística del tiempo de ejecución y el resultado del programa se escribe en un archivo de texto.

Tiempo:
real 0m0.008s
user 0m0.008s
sys 0m0.000s

En el caso del programa secuencial, como resultado de la ejecución del programa podremos ver en pantalla la llave donde el texto cifrado predeterminado es igual al texto cifrado obtenido.

Para ejecutar el programa paralelo en Linux, desde consola se introduce el siguiente comando; al final nos dará la estadística del tiempo de ejecución y el resultado del programa se escribe en un archivo de texto.

fabiola@fabiola-desktop:~$ time ./DES_bitP >tP.txt

Tiempo:
real 0m0.004s
user 0m0.008s
sys 0m0.000s

En la *figura 4.1* se puede apreciar que el programa utiliza diferentes hilos para ejecutar el programa de cifrado con diferentes llaves, además se puede apreciar que aunque se encuentre la llave correcta el programa seguirá en ejecución, debido a que al trabajar con hilos independientes estos no se pueden comunicar haciendo imposible la posibilidad de informar que la llave correcta fue encontrada.

Resultados:

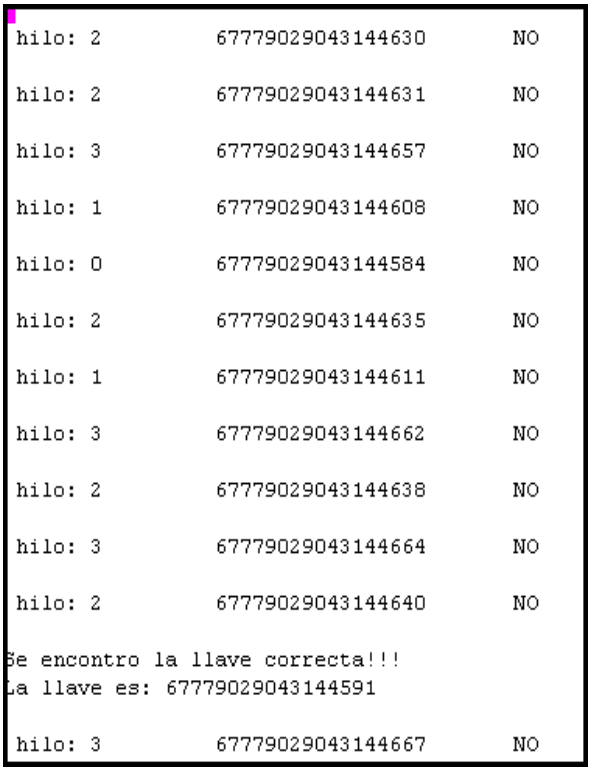


Figura 4.1. Resultado de ejecutar el programa paralelizado.

Se pueden resumir los resultados de algunas pruebas en la siguiente tabla.

Tabla 4.1. Cálculo de tiempos de ejecución en secuencial y paralelo.

Numero de llaves	Tiempo Secuencial (segundos)	Tiempo Paralelo (4 procesadores) (segundos)
100	0.008	0.004
1000	0.054	0.017
10,000	0.502	0.136
100,000	4.973	1.314
1,000,000	49.478	13.142
5,000,000	249.584	65.74
10,000,000	498.438	131.433
20,000,000	989.481	263.198

De la *tabla 4.1*, se puede decir que al comparar los tiempos de ejecución del programa paralelo y el programa secuencial, a medida que el número de datos aumenta, el tiempo necesario para ejecutar el programa con determinado número de llaves del programa paralelo es cada vez más cercano a la cuarta parte del tiempo utilizado en el programa secuencial, lo anterior es fácilmente identificable en la siguiente gráfica.

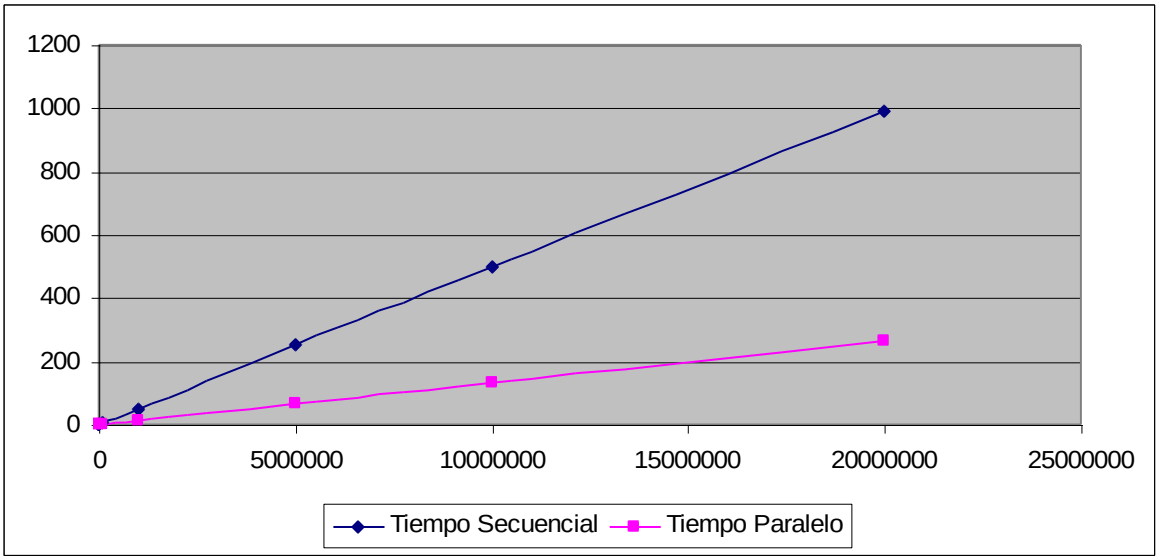


Figura 4.2. Gráfica de la comparación del tiempo de ejecución secuencial y el paralelo

Como se puede apreciar, aunque en ambos casos el tiempo es lineal, la tasa de crecimiento del programa secuencial es más acelerada que la del tiempo paralelo.

4.3 Cálculo de Aceleración (Speedup)

En cómputo paralelo, speedup (aceleración) se refiere a en que proporción es más rápido un algoritmo paralelo que su correspondiente algoritmo secuencial.

Speedup se define por la siguiente formula:

$$S_p = \frac{T_1}{T_p}$$

Donde:

- p es el número de procesadores
- T1 es el tiempo de ejecución de un algoritmo secuencial
- Tp es el tiempo de ejecución del algoritmo paralelo con p procesadores

En el caso del trabajo desarrollado, las pruebas se realizaron con una máquina de 4 procesadores, por lo tanto p=4. Sustituyendo los datos en la fórmula tendremos la siguiente tabla.

Tabla 4.2. Cálculo de aceleración.

Número de llaves	T1	Tp	Sp
100	0.008s	0.004s	2
1000	0.054s	0.017s	3.176
10,000	0.502s	0.136s	3.691
100,000	4.973s	1.314s	3.785
1,000,000	49.478s	13.142s	3.765
5,000,000	249.584s	65.74s	3.796
10,000,000	498.438s	131.433s	3.792
20,000,000	989.481s	263.198s	3.759

De la tabla anterior, graficando los datos se puede apreciar que aunque al número de llaves a probar aumenta, la aceleración se mantiene constante.

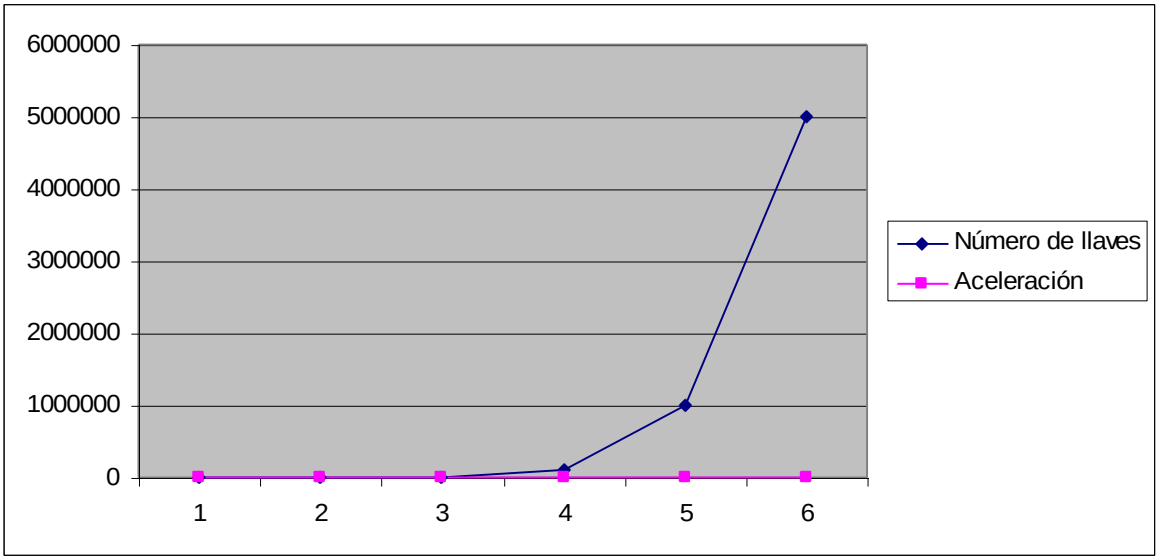


Figura 4.3. Gráfica de aceleración.

La aceleración (speedup) lineal es obtenida cuando Sp = p, es decir, la aceleración es igual al número de procesadores. Al correr un algoritmo con aceleración lineal, doblando el número de procesadores se dobla la velocidad. Es la aceleración ideal y se considera con una muy buena escalabilidad.

Como se puede apreciar en la *gráfica 4.3* la aceleración del programa tiende a ser la ideal, a medida que se prueba con mayor cantidad de llaves, dado que el número de procesadores es 4 y la aceleración es aproximadamente 3.75.

4.4 Eficiencia

La eficiencia es una métrica de desempeño definida como:

$$E_p = \frac{S_p}{p} = \frac{T_1}{pT_p}.$$

Es un valor, entre 0 y 1, que estima que tan bien utilizados están los procesadores al resolver el problema, comparado a que tanto esfuerzo se desperdicia en comunicación y sincronización. Los algoritmos con aceleración lineal y algoritmos

ejecutados en un procesador simple tienen una eficiencia de 1, mientras muchos algoritmos difíciles de paralelizar tienen una eficiencia tal que, 1 / ln(p), que se aproxima a cero a medida que el número de procesadores incrementa.

Al realizar el cálculo de la eficiencia con las muestras tomadas, se obtiene la siguiente tabla.

Número de llaves	Sp	Eficiencia
100	2	0.5
1000	3.176	0.79411765
10,000	3.691	0.92279412
100,000	3.785	0.94615677
1,000,000	3.765	0.94121899
5,000,000	3.796	0.94913295
10,000,000	3.792	0.94808381
20,000,000	3.759	0.93986371

Tabla 4.3. Cálculo de eficiencia.

De la *figura 4.4* anterior, podemos ver que la eficiencia del programa tiende a 1.

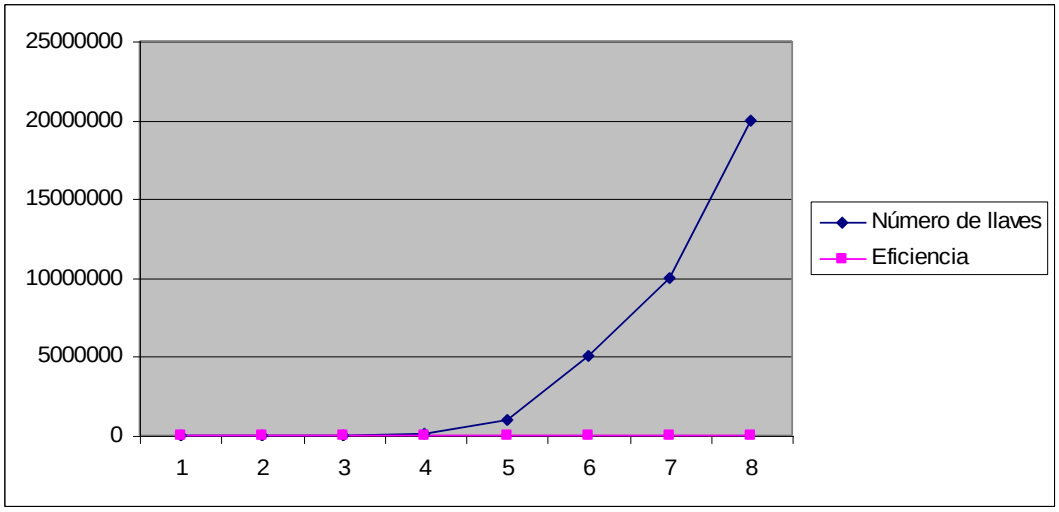


Figura 4.4. Gráfica de eficiencia

CAPÍTULO 5

CONCLUSIONES Y TRABAJOS A FUTURO

5.1 Conclusiones

En el presente trabajo se llevó acabo el desarrollo del ataque por fuerza bruta al algoritmo DES, utilizando la programación paralela en una computadora de múltiples núcleos. Después de realizado el trabajo se ha llegado a las siguientes conclusiones:

- Se diseñó el algoritmo paralelo para realizar el ataque por fuerza bruta a DES, realizando modificaciones al algoritmo de cifrado original DES; eliminando las permutaciones inicial y final, pues aunque no brinda mayor seguridad a la cadena cifrada (debido a que son permutaciones fijas y conocidas), si aumentan el tiempo de ejecución.

- Al programar el algoritmo DES en secuencial se optimizó al máximo el código, trabajando la llave y el mensaje a nivel de bits; usando operaciones, también a nivel de bits (corrimientos, operaciones AND y OR, etc.).

- Al manejar el texto claro y la llave a nivel de bits, fue posible determinar el reacomodo de los elementos de las cajas S, con lo que la sustitución en la aplicación de las cajas S del algoritmo requirió de un menor número de operaciones, dado que se toman los valores de los 6 bits como su equivalente en decimal que indica la posición dentro de la caja utilizada.

- El algoritmo paralelizado se programó utilizando OpenMP. OpenMP permite tener como base un programa en otro lenguaje de programación (C o

Fortran) y modificarlo agregando instrucciones. En la presente investigación, se tomó el código secuencial realizado en lenguaje C y se le incorporaron las instrucciones necesarias para su paralelización, con lo que utilizó los procesadores existentes en la computadora de múltiples núcleos donde es posible probar varias llaves a la vez. En el caso de la computadora de prueba, 4 llaves a la vez. Es posible una rápida modificación del código paralelo en caso de que se utilice una computadora con más o menos núcleos.

- Después de evaluar los programas en secuencial y paralelo, se determinó que la aceleración del programa paralelizado es aproximadamente de 3.7 segundos. En promedio, al comparar los tiempos de ejecución de los programas secuencial y paralelo se puede notar que con 4 procesadores la velocidad de búsqueda de la llave, es aproximadamente 4 veces más rápida.

- Debido a que el programa en paralelo corre sobre hilos, presenta la desventaja de que, aunque la llave correcta se encuentre, el programa continuará su ejecución debido a que cada hilo es independiente y no tiene forma de comunicar a los otros hilos que la llave a sido encontrada, por lo que el programa continuará su ejecución hasta el fin del proceso.

- Aunque se mejoró la velocidad de ejecución y dado que sólo se tienen cuatro procesadores trabajando a la vez, podemos decir que el ataque por fuerza bruta al algoritmo DES utilizando una máquina con estas características no es viable, puesto que tardará aproximadamente más de cien mil años probando todo el universo de llaves.

5.2 Trabajos a futuro

Con base a los puntos mencionados anteriormente y al desarrollo de esta tesis, se abren posibilidades de trabajos a futuro ya sea en este tema así como en temas similares, es decir, implementando un ataque por fuerza bruta al algoritmo DES utilizando programación paralela.

- Basado en el programa secuencial presentado en este trabajo, se puede crear un programa paralelo en otro lenguaje de programación que soporte el trabajo en paralelo pero donde al momento de encontrar la llave buscada se pueda abortar el proceso de búsqueda.

- Implementar el algoritmo paralelo utilizando las ventajas de las tarjetas de procesamiento gráfico (GPUs) que cuentan con múltiples procesadores (dependiendo de la tarjeta 700 ó más procesadores) y actualmente pueden ser implementados aplicaciones de uso general.

APÉNDICE I

CÓDIGO PROGRAMA SECUENCIAL

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/io.h>
#include <ctype.h>
#include <omp.h>


 /* Variables Globales */
   char TxtP[8];
   char TxtC1[4];
   char TxtC2[4];
   char Li[4];
   char Ri[4];
   char aux48[6];
   char aux48Xor[6];
   char aux32[4];
   char aux32_P[4];
   char llavepc1[7];
   char llavepc2[6];

   short tabla_E[48]={31, 0, 1, 2, 3, 4,
                      3, 4, 5, 6, 7, 8,
                      7, 8, 9,10,11,12,
                     11,12,13,14,15,16,
                     15,16,17,18,19,20,
                     19,20,21,22,23,24,
                     23,24,25,26,27,28,
                     27,28,29,30,31, 0};
```

```
short tabla_pc2[48]={13,16,10,23,0 ,4 ,
                     2,27,14,5 ,20,9 ,
                    22,18,11,3 ,25,7 ,
                          15,6 ,26,19,12,1 ,
                          40,51,30,36,46,54,
                          29,39,50,44,32,47,
                          43,48,38,55,33,52,
                          45,41,49,35,28,31};

short tabla_P[32]={15, 6,19,20,
                   28,11,27,16,
                    0,14,22,25,
                    4,17,30, 9,
                    1, 7,23,13,
                   31,26, 2, 8,
                   18,12,29, 5,
                   21,10, 3,24};

char S1_pos[64]={14, 0, 4,15,13, 7, 1, 4, 2,14,15, 2,11,13, 8, 1,
                  3,10,10, 6, 6,12,12,11, 5, 9, 9, 5, 0, 3, 7, 8,
                  4,15,       1,12,14, 8, 8, 2,13, 4, 6, 9, 2, 1,11, 7,
                 15, 5,12,11, 9, 3, 7,14, 3,10,10, 0, 5, 6, 0,13};

char S2_pos[64]={15, 3, 1,13, 8, 4,14, 7, 6,15,11, 2, 3, 8, 4,14,
                  9,12, 7, 0, 2, 1,13,10,12, 6, 0, 9, 5,11,10, 5,
                  0,13,14, 8, 7,10,11, 1,10, 3, 4,15,13, 4, 1, 2,
                  5,11, 8, 6,12, 7, 6,12, 9, 0, 3, 5, 2,14,15, 9};

char S3_pos[64]={10,13, 0, 7, 9, 0,14, 9, 6, 3, 3, 4,15, 6, 5,10,
                  1, 2,13, 8,12, 5, 7,14,11,12, 4,11, 2,15, 8, 1,
                 13, 1, 6,10, 4,13, 9, 0, 8, 6,15, 9, 3, 8, 0, 7,
                 11, 4, 1,15, 2,14,12, 3, 5,11,10, 5,14, 2, 7,12};
```

```
char S4_pos[64]={ 7,13,13, 8,14,11, 3, 5, 0, 6, 6,15, 9, 0,10, 3,
                  1, 4, 2, 7, 8, 2, 5,12,11, 1,12,10, 4,14,15, 9,
                 10, 3, 6,15, 9, 0, 0, 6,12,10,11, 1, 7,13,13, 8,
                 15, 9, 1, 4, 3, 5,14,11, 5,12, 2, 7, 8, 2, 4,14};

char S5_pos[64]={ 2,14,12,11, 4, 2, 1,12, 7, 4,10, 7,11,13, 6, 1,
                  8, 5, 5, 0, 3,15,15,10,13, 3, 0, 9,14, 8, 9, 6,
                  4,11, 2, 8, 1,12,11, 7,10, 1,13,14, 7, 2, 8,13,
                 15, 6, 9,15,12, 0, 5, 9, 6,10, 3, 4, 0, 5,14, 3};

char S6_pos[64]={12,10, 1,15,10, 4,15, 2, 9, 7, 2,12, 6, 9, 8, 5,
                  0, 6,13, 1, 3,13, 4,14,14, 0, 7,11, 5, 3,11, 8,
                  9, 4,14, 3,15, 2, 5,12, 2, 9, 8, 5,12,15, 3,10,
                  7,11, 0,14, 4, 1,10, 7, 1, 6,13, 0,11, 8, 6,13};

char S7_pos[64]={ 4,13,11, 0, 2,11,14, 7,15, 4, 0, 9, 8, 1,13,10,
                  3,14,12, 3, 9, 5, 7,12, 5, 2,10,15, 6, 8, 1, 6,
                  1, 6, 4,11,11,13,13, 8,12, 1, 3, 4, 7,10,14, 7,
                 10, 9,15, 5, 6, 0, 8,15, 0,14, 5, 2, 9, 3, 2,12};

char S8_pos[64]={13, 1, 2,15, 8,13, 4, 8, 6,10,15, 3,11, 7, 1, 4,
                 10,12, 9, 5, 3, 6,14,11, 5, 0, 0,14,12, 9, 7, 2,
                  7, 2,11, 1, 4,14, 1, 7, 9, 4,12,10,14, 8, 2,13,
                  0,15, 6,12,10, 9,13, 0,15, 3, 3, 5, 5, 6, 8,11};
 /***********************PROTOTIPO DE LAS FUNCIONES ***************************/

void genera_llave56(unsigned long long ent);
 void inicializa_textos();
 void inicializa_LiRi();
 void inicializa_llavepc1();
 void corrimientos(short num);
 void aplica_pc2();
 void imprime_cont(int lon, char Vec[]);
void llave_ronda(int ronda);
```

```
 void aplica_E();
 void aplica_Xor(int tam, char dest[], char op[]);
 void aplica_P();
 void aplica_cajas();
 void inicializa_llaveCorrecta();
 int compara_cadbb(int lon, char Vec1[],char Vec2[]);
/*********************************FUNCIONES************************************/
void inicializa_llavepc1()
{
    short i;
    for(i=0;i<7;i++)
       llavepc1[i]=0x0;
}

 void genera_llave56(unsigned long long ent){
     short c_u=0,i,j,bit=0;
     //se inicializa la llavepc1 a 0 antes de leer los bits del numero correspondiente.
     inicializa_llavepc1();
     for (i=6;i>=0;i--){
         for(j=0;j<=7;j++){
            c_u= ent>>bit;   //Se obtiene el primer bit
            c_u&=1;
            c_u<<=j;
            llavepc1[i]|=c_u;  //Se inserta el bit en su lugar correspondiente de la llave
            bit++;
         }
     }
 }

 void corrimientos(short num){
    char c_u[7];
    char aux=0x0;
    short i=0,m=0,n=0;
    for(i=0;i<7;i++)
```

```
  c_u[i]=0x0;
if(num==1){
  for(m=0;m<6;m++)
  {
    c_u[m]=llavepc1[m+1]>>7;
    c_u[m]&=1;//Se eliminan los bits a la izquierda que no se utilizan
  }
  //Bit 0 de la llave[0] a c_u[3] pos 3
  aux=llavepc1[0]>>7;
  aux&=1;
  aux<<=4;
  c_u[3]|=aux;

  //Bit 4 de la llave[3] a c_u[6] pos 7
  c_u[6]=llavepc1[3]>>3;
  c_u[6]&=1;
}
if(num==2){
  for(m=0;m<6;m++)
  {
    c_u[m]=llavepc1[m+1]>>7;
    c_u[m]&=1;//Se eliminan los bits a la izquierda que no se utilizan
    c_u[m]<<=1; //Se recorre a la siguiente posición (6) para dar cabida al siguiente bit

    //Se toma el segundo y se coloca en el último bit de c_u
    aux=llavepc1[m+1]>>6;
    aux&=1;
    c_u[m]|=aux;
  }
  //Bits 4, 5 de la llave[3] a c_u[6] pos 6 y 7
  c_u[6]=llavepc1[3]>>3;
  c_u[6]&=1;
  c_u[6]<<=1;
```

```
      aux=llavepc1[3]>>2;
      aux&=1;
      c_u[6]|=aux;

      //Bits 1,0 de la llave[0] a c_u[3] pos 2 y 3
      aux=llavepc1[0]>>7;
      aux&=1;
      aux<<=5;
      c_u[3]|=aux;

      aux=llavepc1[0]>>6;
      aux&=1;
      aux<<=4;
      c_u[3]|=aux;
   }
   //Se realiza el corrimiento en toda la llave
   for(i=0;i<7;i++)
      llavepc1[i]<<=num;
   //Se hacen 0 los bits 2 y 3 de la llavepc1[3]
   if(num==2)
      llavepc1[3]&=0xCF;
   //Se hace 0 el bit 3 de la llavepc1[3]
   if(num==1)
      llavepc1[3]&=0xEF;
   //Se insertan los bits guardados al final de la llave ya con corrimiento.
   for(i=0;i<7;i++)
      llavepc1[i]|=c_u[i];
 }

void aplica_pc2()
{
    int i,j,k;
    char aux;
    //Se inicializa la llavepc2 a ceros
```

```
    for(i=0;i<6;i++)
      llavepc2[i]=0x0;
    for(i=0;i<48;i++){
      j=tabla_pc2[i]/8;
      k=tabla_pc2[i]%8;
      aux=llavepc1[j]>>(7-k);
      aux&=1;
      j=i/8;
      k=i%8;
      aux<<=7-k;
      llavepc2[j]|=aux;
    }
}

void inicializa_textos()
{
  TxtP[0]=0xCC;
  TxtP[1]=0x0;
  TxtP[2]=0xCC;
  TxtP[3]=0xFF;
  TxtP[4]=0xF0;
  TxtP[5]=0xAA;
  TxtP[6]=0xF0;
  TxtP[7]=0xAA;

  TxtC1[0]=0xA;
  TxtC1[1]=0x4C;
  TxtC1[2]=0xD9;
  TxtC1[3]=0x95;
  TxtC2[0]=0x43;
  TxtC2[1]=0x42;
  TxtC2[2]=0x32;
  TxtC2[3]=0x34;
}
```

```
void inicializa_LiRi()
{
   short i;
   for(i=0;i<4;i++){
      Li[i]=TxtP[i];
      Ri[i]=TxtP[i+4];
   }
}

void imprime_cont(int lon, char Vec[])
{
   short c_u=0,i,j,nb; //nb es el numero de bloques
   nb=lon/8;
   for (i=0;i<nb;i++){
       for (j=7;j>=0;j--){
           c_u= Vec[i]>>j;
           c_u&=1;
           printf("%d",c_u);
       }
       printf("\t");
    }
}

int compara_cadbb(int lon, char Vec1[],char Vec2[])
{
   short c_u1=0,c_u2=0,i,j,nb; //nb es el numero de bloques
   int cont=0;
   nb=lon/8;
   for (i=0;i<nb;i++){
       for (j=7;j>=0;j--){
           c_u1= Vec1[i]>>j;
           c_u1&=1;
           c_u2= Vec2[i]>>j;
```

```
            c_u2&=1;
            if(c_u1!=c_u2)
              break;
            else
              cont++;
        }
      }
      if(lon==cont)
        return 1;
      else
        return 0;
}

void llave_ronda(int ronda)
{
   int c;
   if (ronda == 0|ronda == 1|ronda == 8|ronda == 15) //Rondas de 0 a 15
     c=1;
   else
     c=2;
   corrimientos(c);
   aplica_pc2();
}

void aplica_E(){
     int i,j,k;
     char aux;
     //Se inicializa aux48
     for(i=0;i<6;i++)
        aux48[i]=0x0;
     for(i=0;i<48;i++){
        j=tabla_E[i]/8;
        k=tabla_E[i]%8;
        aux=Ri[j]>>(7-k);
```

```
        aux&=1;
        j=i/8;
        k=i%8;
        aux<<=7-k;
        aux48[j]|=aux;
    }
}

void aplica_P(){
    int i,j,k;
    char aux;
    //Se inicializa aux32_P
    for(i=0;i<4;i++)
        aux32_P[i]=0x0;  //16
    for(i=0;i<32;i++){
        j=tabla_P[i]/8;
        k=tabla_P[i]%8;
        aux=aux32[j]>>(7-k);
        aux&=1;
        j=i/8;
        k=i%8;
        aux<<=7-k;
        aux32_P[j]|=aux;
    }
}

void aplica_Xor(int tam, char dest[], char op[]){
    int i;
    int tope;
    tope=tam/8;
    for(i=0;i<tope;i++){
        dest[i]^=op[i];
    }
}
```

```
void aplica_cajas(){
  int i,j=0,k=0,p=0,i_aux32=0, j_aux32=0, k_aux32=0,l; //j posicion del vector, k posición de bit y p es el paquete
  int pos_ca=0;
  char aux1=0x0, aux2, aux3=0x0;
  for(i=0;i<4;i++)
    aux32[i]=0x0;
  for(p=0;p<8;p++){
    i=0;
    aux2=0x0;
    while((i<6)&&(p<8)){ //i cuenta los bits a obtener (paquetes de 6)
      aux1=aux48[j]>>(7-k);
      aux1&=1;
      aux1<<=7-i-2;
      aux2|=aux1;
      if(k==7){
        k=0;
        j++;
      }else
        k++;
      i++;
    }
    pos_ca=aux2; //Posicion de la caja correspondiente
    switch (p){
      case 0: //En la primera ronda se usa SBox1
            aux3=S1_pos[pos_ca];
              break;
        case 1: //El la segunda ronda se usa SBox2
            aux3=S2_pos[pos_ca];
              break;
      case 2: //El la tercera ronda se usa SBox3
            aux3=S3_pos[pos_ca];
                break;
```

```
            case 3: //El la cuarta ronda se usa SBox4
                aux3=S4_pos[pos_ca];
                    break;
            case 4: //El la quinta ronda se usa SBox5
                aux3=S5_pos[pos_ca];
                        break;
            case 5: //El la sexta ronda se usa SBox6
                aux3=S6_pos[pos_ca];
                    break;
            case 6: //El la séptima ronda se usa SBox7
                        aux3=S7_pos[pos_ca];
                        break;
            case 7: //El la octava ronda se usa SBox8
                aux3=S8_pos[pos_ca];
                    break;
            default:
                    break;
        }///switch
        pos_ca=0;
        for(l=0;l<4;l++){
            aux1=aux3>>(3-l);
            aux1&=1;
            j_aux32=i_aux32/8;
            k_aux32=i_aux32%8;
            aux1<<=7-k_aux32;
            aux32[j_aux32]|=aux1;
            i_aux32++;
        }//for0-3
    }//for
}

int main(){
    unsigned long long t_k,t_ki,a3;
    int i,l,correcto=1;
```

```
  long long int llaves;
  int tx1,tx2;
  long a1,a2;
  inicializa_textos();
  t_ki=67779029043140000;//--llave correcta--67779029043144591
  for(llaves=0;llaves<100000;llaves++) {
    inicializa_LiRi();
    t_k=t_ki+llaves;
    genera_llave56(t_k);
     //Ciclo que realiza las rondas del procedimiento
    for(i=0;i<16;i++) {
      llave_ronda(i); //Ki estaria en llavepc2
      aplica_E();
      aplica_Xor(48, aux48, llavepc2);
      aplica_cajas();
      aplica_P();
      aplica_Xor(32,aux32_P,Li);
      //Se prepara a Li y Ri para la siguiente ronda
      for(l=0;l<4;l++){
        Li[l]=Ri[l];
        Ri[l]=aux32_P[l];
      }
    }
    tx1=compara_cadbb(32, Ri,TxtC1);
    tx2=compara_cadbb(32, Li,TxtC2);
    if((tx1==1)&&(tx2==1)) {
        printf("\nSe encontro la llave correcta!!!\n");
        printf("La llave es: %llu, %d\n",t_k,llaves);
    }
    else
      printf("\n %llu\t NO\n",t_k);
 }
  return 1;
}
```

APÉNDICE I

CÓDIGO PROGRAMA PARALELIZADO

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/io.h>
#include <ctype.h>
#include <omp.h>


 /* Variables Globales */
   char TxtP[8];
   char TxtC1[4];
   char TxtC2[4];
   char Li[4];
   char Ri[4];
   char aux48[6];
   char aux48Xor[6];
   char aux32[4];
   char aux32_P[4];
   char llavepc1[7];
   char llavepc2[6];

   short tabla_E[48]={31, 0, 1, 2, 3, 4,
                       3, 4, 5, 6, 7, 8,
                       7, 8, 9,10,11,12,
                      11,12,13,14,15,16,
                      15,16,17,18,19,20,
                      19,20,21,22,23,24,
                      23,24,25,26,27,28,
                      27,28,29,30,31, 0};
```

```
//Tabla considerada a partir del bit 0 y no del 1 (como la original)
short tabla_pc2[48]={13,16,10,23,0 ,4 ,
                     2,27,14,5 ,20,9 ,
                    22,18,11,3 ,25,7 ,
                          15,6 ,26,19,12,1 ,
                          40,51,30,36,46,54,
                          29,39,50,44,32,47,
                          43,48,38,55,33,52,
                          45,41,49,35,28,31};

short tabla_P[32]={15, 6,19,20,
                   28,11,27,16,
                    0,14,22,25,
                    4,17,30, 9,
                    1, 7,23,13,
                   31,26, 2, 8,
                   18,12,29, 5,
                   21,10, 3,24};

char S1_pos[64]={14, 0, 4,15,13, 7, 1, 4, 2,14,15, 2,11,13, 8, 1,
                 3,10,10, 6, 6,12,12,11, 5, 9, 9, 5, 0, 3, 7, 8,
                 4,15,       1,12,14, 8,     8, 2,13, 4,       6, 9, 2, 1,11, 7,
                15, 5,12,11, 9, 3,  7,14, 3,10,10, 0, 5, 6, 0,13};

char S2_pos[64]={15, 3, 1,13, 8, 4,14, 7, 6,15,11, 2, 3, 8, 4,14,
                 9,12, 7, 0, 2, 1,13,10,12, 6, 0, 9, 5,11,10, 5,
                 0,13,14, 8, 7,10,11, 1,10, 3, 4,15,13, 4, 1, 2,
                 5,11, 8, 6,12, 7, 6,12, 9, 0, 3, 5, 2,14,15, 9};

char S3_pos[64]={10,13, 0, 7, 9, 0,14, 9, 6, 3, 3, 4,15, 6, 5,10,
                 1, 2,13, 8,12, 5, 7,14,11,12, 4,11, 2,15, 8, 1,
                13, 1, 6,10, 4,13, 9, 0, 8, 6,15, 9, 3, 8, 0, 7,
```

```
        11, 4, 1,15, 2,14,12, 3, 5,11,10, 5,14, 2, 7,12};

char S4_pos[64]={ 7,13,13, 8,14,11, 3, 5, 0, 6, 6,15, 9, 0,10, 3,
        1, 4, 2, 7, 8, 2, 5,12,11, 1,12,10, 4,14,15, 9,
        10, 3, 6,15, 9, 0, 0, 6,12,10,11, 1, 7,13,13, 8,
        15, 9, 1, 4, 3, 5,14,11, 5,12, 2, 7, 8, 2, 4,14};

char S5_pos[64]={ 2,14,12,11, 4, 2, 1,12, 7, 4,10, 7,11,13, 6, 1,
        8, 5, 5, 0, 3,15,15,10,13, 3, 0, 9,14, 8, 9, 6,
        4,11, 2, 8, 1,12,11, 7,10, 1,13,14, 7, 2, 8,13,
        15, 6, 9,15,12, 0, 5, 9, 6,10, 3, 4, 0, 5,14, 3};

char S6_pos[64]={12,10, 1,15,10, 4,15, 2, 9, 7, 2,12, 6, 9, 8, 5,
        0, 6,13, 1, 3,13, 4,14,14, 0, 7,11, 5, 3,11, 8,
        9, 4,14, 3,15, 2, 5,12, 2, 9, 8, 5,12,15, 3,10,
        7,11, 0,14, 4, 1,10, 7, 1, 6,13, 0,11, 8, 6,13};

char S7_pos[64]={ 4,13,11, 0, 2,11,14, 7,15, 4, 0, 9, 8, 1,13,10,
        3,14,12, 3, 9, 5, 7,12, 5, 2,10,15, 6, 8, 1, 6,
        1, 6, 4,11,11,13,13, 8,12, 1, 3, 4, 7,10,14, 7,
        10, 9,15, 5, 6, 0, 8,15, 0,14, 5, 2, 9, 3, 2,12};

char S8_pos[64]={13, 1, 2,15, 8,13, 4, 8, 6,10,15, 3,11, 7, 1, 4,
        10,12, 9, 5, 3, 6,14,11, 5, 0, 0,14,12, 9, 7, 2,
        7, 2,11, 1, 4,14, 1, 7, 9, 4,12,10,14, 8, 2,13,
        0,15, 6,12,10, 9,13, 0,15, 3, 3, 5, 5, 6, 8,11};
 /**********************PROTOTIPO DE LAS FUNCIONES ***************************/

void genera_llave56(unsigned long long ent);
 void inicializa_textos();
 void inicializa_LiRi();
 void inicializa_llavepc1();
 void corrimientos(short num);
 void aplica_pc2();
```

```
 void imprime_cont(int lon, char Vec[]);
 void llave_ronda(int ronda);
 void aplica_E();
 void aplica_Xor(int tam, char dest[], char op[]);
 void aplica_P();
 void aplica_cajas();
 void inicializa_llaveCorrecta();
 int compara_cadbb(int lon, char Vec1[],char Vec2[]);
/*********************************FUNCIONES************************************/
void inicializa_llavepc1()
{
    short i;
    for(i=0;i<7;i++)
       llavepc1[i]=0x0;
}

 void genera_llave56(unsigned long long ent){
     short c_u=0,i,j,bit=0;
     inicializa_llavepc1();
     for (i=6;i>=0;i--){
         for(j=0;j<=7;j++){
            c_u= ent>>bit;
            c_u&=1;
            c_u<<=j;
            llavepc1[i]|=c_u;
            bit++;
         }
     }
 }

void corrimientos(short num){
    char c_u[7];
    char aux=0x0;
    short i=0,m=0,n=0;
```

```
for(i=0;i<7;i++)
  c_u[i]=0x0;
if(num==1){
  for(m=0;m<6;m++) {
    c_u[m]=llavepc1[m+1]>>7;
    c_u[m]&=1;//Se eliminan los bits a la izquierda que no se utilizan
  }
  //Bit 0 de la llave[0] a c_u[3] pos 3
  aux=llavepc1[0]>>7;
  aux&=1;
  aux<<=4;
  c_u[3]|=aux;
  //Bit 4 de la llave[3] a c_u[6] pos 7
  c_u[6]=llavepc1[3]>>3;
  c_u[6]&=1;
}
if(num==2){
  for(m=0;m<6;m++) {
    c_u[m]=llavepc1[m+1]>>7;
    c_u[m]&=1;//Se eliminan los bits a la izquierda que no se utilizan
    c_u[m]<<=1;
    aux=llavepc1[m+1]>>6;
    aux&=1;
    c_u[m]|=aux;
  }
  //Bits 4, 5 de la llave[3] a c_u[6] pos 6 y 7
  c_u[6]=llavepc1[3]>>3;
  c_u[6]&=1;
  c_u[6]<<=1;
  aux=llavepc1[3]>>2;
  aux&=1;
  c_u[6]|=aux;
  //Bits 1,0 de la llave[0] a c_u[3] pos 2 y 3
  aux=llavepc1[0]>>7;
```

```
      aux&=1;
      aux<<=5;
      c_u[3]|=aux;
      aux=llavepc1[0]>>6;
      aux&=1;
      aux<<=4;
      c_u[3]|=aux;
   }
   //Se realiza el corrimiento en toda la llave
   for(i=0;i<7;i++)
      llavepc1[i]<<=num;
   //Se hacen 0 los bits 2 y 3 de la llavepc1[3]
   if(num==2)
      llavepc1[3]&=0xCF;
   if(num==1)
      llavepc1[3]&=0xEF;
   for(i=0;i<7;i++)
      llavepc1[i]|=c_u[i];
}

void aplica_pc2()
{
    int i,j,k;
    char aux;
    //Se inicializa la llavepc2 a ceros
    for(i=0;i<6;i++)
       llavepc2[i]=0x0;
    for(i=0;i<48;i++){
       j=tabla_pc2[i]/8;
       k=tabla_pc2[i]%8;
       aux=llavepc1[j]>>(7-k);
       aux&=1;
       j=i/8;
       k=i%8;
```

```
        aux<<=7-k;
        llavepc2[j]|=aux;
    }
}

//Función que inicializa las variables Lo, Ro, Lc, Rc. Para comenzar con el proceso.
void inicializa_textos()
{
  //Inicialización del texto original despues de Permutacion Inicial
  TxtP[0]=0xCC;
  TxtP[1]=0x0;
  TxtP[2]=0xCC;
  TxtP[3]=0xFF;
  TxtP[4]=0xF0;
  TxtP[5]=0xAA;
  TxtP[6]=0xF0;
  TxtP[7]=0xAA;
  //Inicialización del texto cifrado (texto antes de aplicar IP-1)
  TxtC1[0]=0xA;
  TxtC1[1]=0x4C;
  TxtC1[2]=0xD9;
  TxtC1[3]=0x95;
  TxtC2[0]=0x43;
  TxtC2[1]=0x42;
  TxtC2[2]=0x32;
  TxtC2[3]=0x34;
}

void inicializa_LiRi()
{
    short i;
    for(i=0;i<4;i++){
       Li[i]=TxtP[i];
       Ri[i]=TxtP[i+4];
```

```
    }
}

void imprime_cont(int lon, char Vec[])
{
    short c_u=0,i,j,nb; //nb es el numero de bloques
    nb=lon/8;
    for (i=0;i<nb;i++){
        for (j=7;j>=0;j--){
            c_u= Vec[i]>>j;
            c_u&=1;
            printf("%d",c_u);
        }
        printf("\t");
    }
}

int compara_cadbb(int lon, char Vec1[],char Vec2[])
{
    short c_u1=0,c_u2=0,i,j,nb; //nb es el numero de bloques
    int cont=0;
    nb=lon/8;
    for (i=0;i<nb;i++){
        for (j=7;j>=0;j--){
            c_u1= Vec1[i]>>j;
            c_u1&=1;
            c_u2= Vec2[i]>>j;
            c_u2&=1;
            if(c_u1!=c_u2)
              break;
            else
              cont++;
        }
    }
```

```
    //printf("En compara %d=%d", lon,cont);
    if(lon==cont)
      return 1;
    else
      return 0;
}

void llave_ronda(int ronda)
{
  int c;
  if (ronda == 0|ronda == 1|ronda == 8|ronda == 15) //Rondas consideradas de 0 a 15
    c=1;
  else
    c=2;
  corrimientos(c);
  aplica_pc2();
}

void aplica_E(){
    int i,j,k;
    char aux;
    //Se inicializa aux48
    for(i=0;i<6;i++)
      aux48[i]=0x0;
    for(i=0;i<48;i++){
      j=tabla_E[i]/8;
      k=tabla_E[i]%8;
      aux=Ri[j]>>(7-k);
      aux&=1;
      j=i/8;
      k=i%8;
      aux<<=7-k;
      aux48[j]|=aux;
    }
```

```
}

void aplica_P(){
    int i,j,k;
    char aux;
    for(i=0;i<4;i++)
      aux32_P[i]=0x0;  //16
    for(i=0;i<32;i++){
      j=tabla_P[i]/8;
      k=tabla_P[i]%8;
      aux=aux32[j]>>(7-k);
      aux&=1;
      j=i/8;
      k=i%8;
      aux<<=7-k;
      aux32_P[j]|=aux;
    }
}

void aplica_Xor(int tam, char dest[], char op[]){
    int i;
    int tope;
    tope=tam/8;
    for(i=0;i<tope;i++){
      dest[i]^=op[i];
    }
}


void aplica_cajas(){
  int i,j=0,k=0,p=0,i_aux32=0, j_aux32=0, k_aux32=0,l; //j posicion del vector, k posición de bit y p es el paquete
  int pos_ca=0;
  char aux1=0x0, aux2, aux3=0x0;
```

```
for(i=0;i<4;i++)
  aux32[i]=0x0;
for(p=0;p<8;p++){
  i=0;
  aux2=0x0;
  while((i<6)&&(p<8)){ //i cuenta los bits a obtener (paquetes de 6)
    aux1=aux48[j]>>(7-k);
    aux1&=1;
    aux1<<=7-i-2;
    aux2|=aux1;
    if(k==7){
      k=0;
      j++;
    }else
      k++;
    i++;
  }
  pos_ca=aux2; //Posicion de la caja correspondiente
  switch (p){
     case 0:
          aux3=S1_pos[pos_ca];
               break;
        case 1:
          aux3=S2_pos[pos_ca];
             break;
     case 2:
          aux3=S3_pos[pos_ca];
               break;
        case 3:
          aux3=S4_pos[pos_ca];
             break;
     case 4:
          aux3=S5_pos[pos_ca];
               break;
```

```
            case 5:
                aux3=S6_pos[pos_ca];
                  break;
            case 6:
                      aux3=S7_pos[pos_ca];
                      break;
            case 7:
                aux3=S8_pos[pos_ca];
                  break;
            default:
                  break;
      }///switch
      pos_ca=0;
      for(l=0;l<4;l++){
        aux1=aux3>>(3-l);
        aux1&=1;
        j_aux32=i_aux32/8;
        k_aux32=i_aux32%8;
        aux1<<=7-k_aux32;
        aux32[j_aux32]|=aux1;
        i_aux32++;
      }//for0-3
    }//for
}

int main(){
  unsigned long long t_k,t_ki,a3;
  int i,l,correcto=1;
  //int llaves;
  long long int llaves;
  int tx1,tx2;
  int wait, nthread=4;
  long a1,a2;
  inicializa_textos();
```

```
t_ki=67779029043140000;//--llave correcta--67779029043144591
omp_set_num_threads(nthread);
#pragma omp parallel for
  for(llaves=0;llaves<100000;llaves++) {
    #pragma omp sections {
      inicializa_LiRi();
      t_k=t_ki+llaves;
      genera_llave56(t_k);
      for(i=0;i<16;i++) {
        llave_ronda(i); //Ki estaran llavepc2
        aplica_E();
        aplica_Xor(48, aux48, llavepc2);
        aplica_cajas();
        aplica_P();
        aplica_Xor(32,aux32_P,Li);
        for(l=0;l<4;l++){
          Li[l]=Ri[l];
          Ri[l]=aux32_P[l];
        }
      }
      tx1=compara_cadbb(32, Ri,TxtC1);
      tx2=compara_cadbb(32, Li,TxtC2);
      if((tx1==1)&&(tx2==1)) {
        printf("\nSe encontro la llave correcta!!!\n");
        printf("La llave es: %llu\n",t_k);
      }
      else
        printf("\n hilo: %d\t %llu\t NO\n",omp_get_thread_num(),t_k);
    }//fin sections
  }//fin for
}
```

Referencias

[1] Silva García V. M., “Criptoanálisis para la modificación de los estándares DES y Triple DES”, Tesis de Doctorado, México, D.F. 2007.

[2] Libro electrónico: “Aplicaciones Criptográficas”, 2ª Edición, 1999, ISBN 83-87238-57-2, publicado por el departamento de Publicaciones de la Escuela Universitaria de Información de la Universidad Politécnica de Madrid, España. http://www.criptored.upm.es/guiateoria/gt_m001a.htm

[3] Fuster Sabater A. et al. “Técnicas Criptográficas de protección de datos”, 2ª Edición, Alfaomega 2001.

[4] Chapman, Barbara et al. “Using OpenMP. Portable shared memory parallel programming”. The MIT Press. Cambridge. 2008.

[5] “Historia de la criptografía” Reportaje por José Luis Vratny. Disponible en: http://www.latinoseguridad.com/LatinoSeguridad/Reps/Cripto1.shtml

[6] Ritter T, 2006, “Triple-DES is Proven to be Very Secure?”, http://www.ciphersbyritter.com/NEWS5/PROVSEC.HTM

[7] Arriazu S., Jorge. “Descripción del Algoritmo DES (Data Encryption Standar)”. Diciembre de 1999.

[8] “Data Encryption Standard” Artículo disponible en: http://es.wikipedia.org

[9] RSA Data Security. RSA Laboratories Secret-Key Challenge. http://www.rsa.com/rsalabs/node.asp?id=2091

[10] Cracking DES. Secrets of how federal encryption research agencies wiretap politics. Electronic Frontier Foundation. United States of America. Public Domain. 1998.

[11] Ghnaim, Wafaa Abd-Elmonim et al, “Known-Ciphertext Cryptanalysis Approach for the Data Encryption Standard Technique.” IEEE. International Conference on Computer Information Systems and Industrial Management Applications (CISIM). pp. 600-603. 2010.

[12] Stamp, Mark. “Information Security. Principles and Practice”. John Wiley & Sons, Inc. 2006.

[13] Douglas R. Stinson. “Cryptography Theory and Practice”. 3er Ed. Chapman & Hall/CRC. pp. 95-102. 2006.

[14] FIPS PUB 46-3. Federal Information Processing Standards Publication. Reaffirmed October 25, 1999. U. S. Department of Commerce / National Institute of Standards and Technology.

[15] Quinn, Michael J. “Parallel Programming in C with MPI and OpenMP”. Mc Graw Hill. 2004.

[16] Leighton F. Thomson, “Introduction to Parallel Algorithms and Architectures: Arrays – Trees - Hypercubes”, Morgan Kaufmann Publishers, Inc. 1992.

[17] “Evolución histórica de los procesadores” http://www.fdi.ucm.es/profesor/sdelpino/ETC/historia.pdf

[18] http://www.intel.com

[19] E. Biham, A. Shamir, “Differential cryptanalysis of DES-like cryptosystems. Extended Abstract.” The Weizmann Institute of Science Department of Applied Mathematics.

[20] M. Matsui. “Linear Cryptanalysis Method for DES Cipher, Advances in Cryptology.”

[21] M. Curtin, J. Dolske. “A Brute Force Search of DES Keyspace”. Mayo, 1998.

[22] http://www.eff.org/descracker/

[23] DES-III: Success. http://www.distributed.net/DES

[24] Zhengjun Cao y Lihua Liu. “An Attack Against DES Based On The Relationship Li+1 = Ri.” Third International Symposium on Electronic Commerce and Security. IEEE Computer Society. pp. 280-283. 2010.

[25] Agosta, Giovanni et al. “Record Setting Software Implementation of DES Using CUDA”. Seventh International Conference on Information Technology. 2010.