ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UN ALGORITMO PARA ROMPER POR FUERZA BRUTA AL SIMPLIFIED DATA ENCRYPTION STANDARD (S-DES) MEDIANTE EL USO DE COMPUTACIÓN PARALELA.

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERA EN ELECTRÓNICA Y REDES DE INFORMACIÓN

VANESSA ALEXANDRA GUEVARA SAMANIEGO

DIRECTOR: Ph.D. ROBIN GERARDO ÁLVAREZ RUEDA

Quito, julio 2019

AVAL

Certifico que el presente trabajo fue desarrollado por Vanessa Alexandra Guevara Samaniego, bajo mi supervisión.

Ph.D. Robin Álvarez
DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo, Vanessa Alexandra Guevara Samaniego, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejo (dejamos) constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.

VANESSA ALEXANDRA GUEVARA SAMANIEGO

DEDICATORIA

Este proyecto está dedicado principalmente a Dios, incondicional, amoroso y misericordioso, por las oportunidades, por todo lo que hace y seguirá haciendo dentro de mi vida, que su voluntad se cumpla.

También se lo dedico a mis padres por su paciencia y apoyo, demostrando que en la vida se puede lograr todo si se lo propone.

Finalmente lo dedico a mis hermanas por ser mi soporte y mi ayuda en todo momento.

Vanessa Alexandra Guevara Samaniego

AGRADECIMIENTO

Agradezco a Dios por regalarme tanto, por acompañarme y bendecirme en todo el camino, porque he visto sus milagros en mi vida, por el estoy aquí.

A mis padres

Por el esfuerzo, paciencia, amor y apoyo en todo momento, por los valores que siempre me enseñaron.

A mis hermanas

Por su ejemplo, amor y apoyo en cada uno de los pasos que me ha permitido llegar aquí, por su comprensión, por ser mi soporte en los momentos difíciles y ser mi alegría en mis momentos felices, por su tiempo, ayuda, por ser incondicionales.

Al Dr Robin Alvarez, por su colaboración, paciencia y apoyo en la realización de este proyecto.

Vanessa Alexandra Guevara Samaniego

ÍNDICE DE CONTENIDO

SIGLAS

- PTC: Parallel Toolbox Computing
- MDCE: MATLAB Distributed Computing Engine.
- MDCS: MATLAB Distributed Computing Server.
- DES: Data Encryption Estandar.
- S-DES: Simplified Data Encryption Estandar.

RESUMEN

Este trabajo presenta la implementación de un algoritmo de ataque por fuerza bruta al Simplified Data Encryption Estándar (S-DES), el cual será ejecutado tanto de manera serial, tradicionalmente empleada, como en forma paralela utilizando para este efecto las herramientas de computación paralela provistas por MATLAB: Parallel Toolbox Computing (PTC) y MATLAB Distributed Computing Server (MDCE). El objetivo de realizar esta paralelización, es reducir el tiempo de evaluación del espacio de claves. En el ataque en forma paralela se incluye tanto la ejecución en un solo equipo, explotando la característica de multinúcleo que se tiene en la actualidad, como la ejecución en 3 equipos que formarán un clúster computacional.

Para facilidad de uso de los usuarios, se implementa una interfaz gráfica de modo que se pueda realizar el cifrado, descifrado y el ataque en forma serial y paralela.

Finalmente, en las pruebas de funcionamiento, se realiza una comparación entre los tiempos de demora de rompimiento del algoritmo en función de la cantidad de recursos computacionales empleados: una sola computadora trabajando en forma serial, en forma paralela y varias computadoras trabajando en forma paralela. Adicionalmente, dichas pruebas se realizan en dos ambientes: en una red cableada y en una red inalámbrica Ad-hoc.

PALABRAS CLAVE: seguridad informática, ataques criptográficos, algoritmos de cifrado-descifrado-ataque, fuerza bruta, computación paralela, clúster.

ABSTRACT

This paper presents the implementation of a brute-force attack algorithm to the Simplified Data Encryption Standard (S-DES), which will be executed both serially, traditionally used, and in parallel using the parallel computing Toolbox provided for this purpose by MATLAB: Parallel Toolbox Computing (PTC) and MATLAB Distributed Computing Server (MDCE). The objective of this parallelization is to reduce the evaluation time of the key space. The parallel attack includes both the execution in a single computer, exploiting the multi-core feature that is currently available, and the execution in 3 computers that will form a computational cluster.

For user's ease of use, a graphical interface is implemented so that encryption, decryption and attack can be performed serially and in parallel.

Finally, in the performance tests, a comparison is made between the breakdown delay times of the algorithm based on the amount of computational resources used: a single computer working serially, in parallel and several computers working in parallel. Additionally, these tests are carried out in two environments: in a wired network and in an Ad-hoc wireless network.

KEYWORDS: computer security, cryptographic attacks, encryption-decryption-attack algorithms, brute force, parallel computing, cluster.

1. INTRODUCCIÓN

Si bien existen una gran cantidad de algoritmos de seguridad, conforme la tecnología avanza, la intromisión ilegal dentro de las comunicaciones también lo hace, pero cuando la información está cifrada, lo obtenido no tendría sentido para el intruso; sin embargo, atacando un algoritmo de cifrado se puede obtener el mensaje en claro, pero ¿cómo se realiza este proceso?, el presente proyecto pretende mostrar una explicación didáctica mediante una aplicación de ataque al algoritmo de seguridad Simplified Data Encryption Estándar (S-DES).

Se centra en un algoritmo de cifrado simétrico, cuyo secreto es únicamente la clave con la que ha sido cifrado, para realiza el ataque por fuerza bruta. Este ataque se puede considerar el más sencillo si se lo compara con el criptoanálisis, el mismo pretende romper un algoritmo analizando su desarrollo y encontrado debilidades, en este caso lo principal será evaluar todas las posibles combinaciones del espacio de claves, este número se incrementa conforme la longitud de la clave lo haga y con ello también aumenta la necesidad de mayores recursos computaciones, para solventar esto se utilizará procesamiento paralelo con el objetivo de optimizar la ejecución de dicha aplicación haciendo un uso eficiente de los multiprocesadores que contienen los computadores partícipes del proceso.

1.1 OBJETIVOS

El objetivo general de este Estudio Técnico es desarrollar un algoritmo para romper por fuerza bruta al Simplified Data Encryption Standard (S-DES) mediante el uso de computación paralela.

Los objetivos específicos de este Estudio Técnico son:

- Analizar el cifrado y descifrado del Simplified Data Encryption Standard(S-DES).
- Diseñar una aplicación que integre los algoritmos cifrado, descifrado y ataque del Simplified Data Encryption Standard.
- Analizar los resultados obtenidos al comparar los tiempos de procesamiento del ataque en cada escenario que se especificará más adelante.

1.2. ALCANCE

Con la finalidad de ofrecer una aplicación de uso didáctico, el presente trabajo pretende implementar los siguientes algoritmos S-DES cifrado /descifrado (figura 1.1) y el ataque sin conocer la clave (figura1.2).

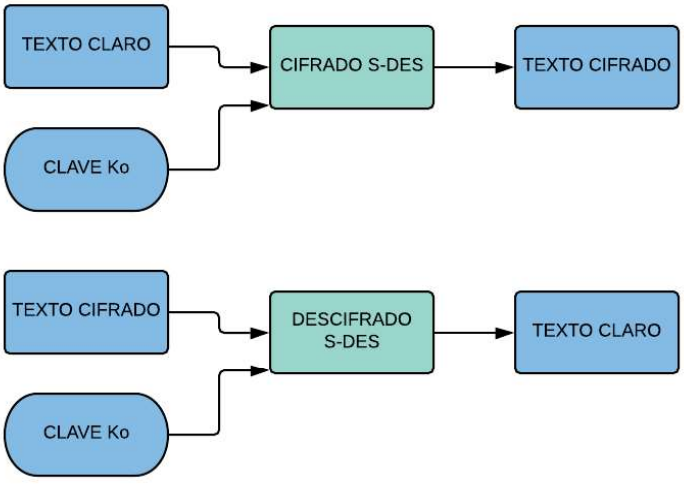


Figura 1.1. **Esquema de cifrado/descifrado con S-DES**

Para implementar los algoritmos y la interfaz gráfica de usuario se utilizará el software Matlab. Los algoritmos se ejecutarán en una sola computadora o en varias, según sea el escenario que se especifica más adelante. Las computadoras participantes trabajarán sobre Windows.

Para la formación del clúster, se utilizará las siguientes herramientas: Matlab Distributed Computing Services (MDCS) y Parallel Computing Toolbox. Para la distribución del trabajo entre núcleos, es el administrador quien divida las porciones de datos.

La aplicación tomará como fuente de información un texto plano (cuya longitud será al menos de 1 página) para que sea cifrado mediante una clave generada aleatoriamente [1], el archivo cifrado será utilizado como entrada al algoritmo de ataque, el cual descubrirá la clave y con esta el archivo original. Se podrá elegir el escenario de ataque, pudiendo ser: secuencial, paralelo con una sola computadora o paralelo con múltiples computadoras como se muestra en la figura 1.2.

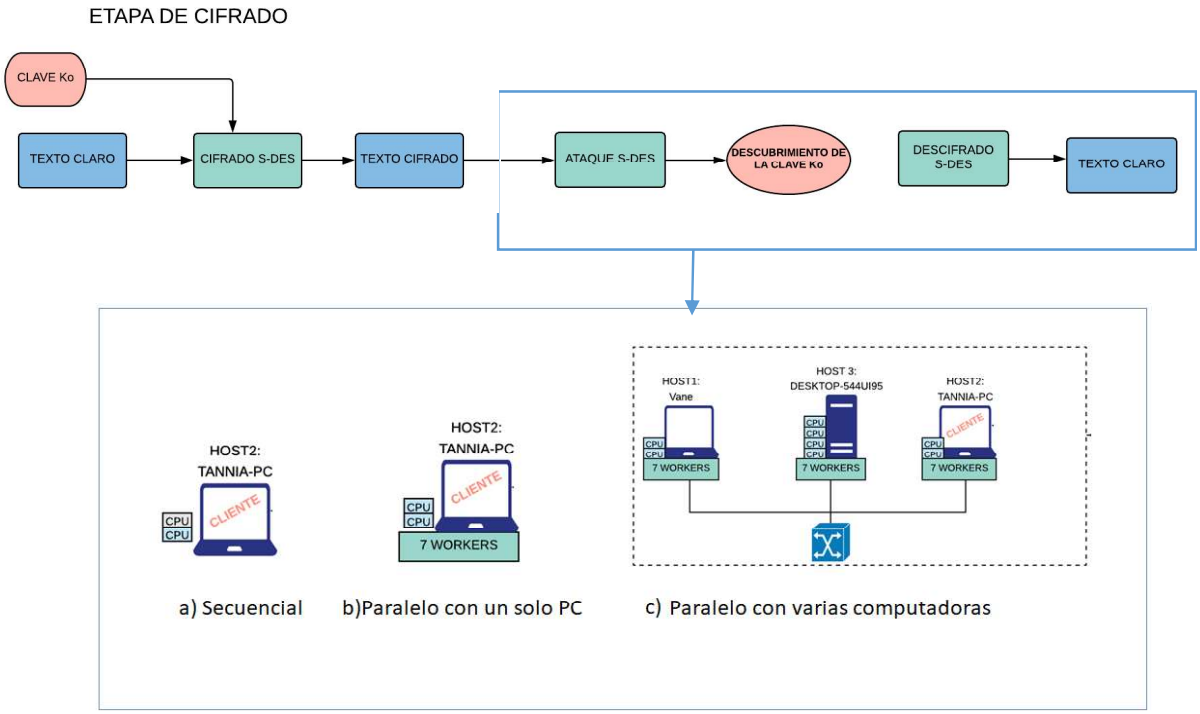


Figura 1.2. Escenarios para el ataque por fuerza bruta al S-DES mediante computación paralela.

A continuación, se describen los escenarios a implementar:

- **Secuencial:** Se ejecutará el algoritmo en una sola computadora de manera serial.
- **Paralelo en una sola computadora**: Se ejecutará el algoritmo de ataque (con ciertas modificaciones para permitir el paralelismo) mediante la utilización de los múltiples núcleos de una computadora.
- **Paralelo en varias computadoras:** Se ejecutará el algoritmo de ataque utilizando los múltiples núcleos de 3 computadoras conectadas en red.
  Dentro de este se consideran 2 sub escenarios, el primero mediante red cableada y el segundo mediante conexión inalámbrica.

1.3 MARCO TEÓRICO

1.3.1. SEGURIDAD INFORMÁTICA Y CRIPTOGRAFÍA

El Comité de Sistemas de Seguridad Nacional (CNSS) puntualiza la seguridad de la información como la protección de la información y sus elementos críticos, incluidos el software y el hardware que utiliza, almacena y transmite esa información.

En palabras más sencillas, se puede definir a la seguridad de la información como un conjunto de herramientas que permiten la protección de la información de cualquier amenaza.

En el estándar ISO 7498-2 [2], se establece una arquitectura de seguridad con el objetivo de proteger los servicios de comunicaciones que son los siguientes:

- **Confidencialidad**: Consiste en que la información solo podrá ser utilizada por las personas o equipos autorizados, para avalar esta función se ayuda de la autenticación, autorización y el cifrado.
- **Integridad**: Asegura que los datos no sean alterados por un tercero
- **Autenticación de emisor** Este proceso consiste en una confirmación de la identidad de una persona o equipo, es decir comprobación de que es quien dice ser y no un intruso.
- **No repudio**: certifica la participación de las entidades comunicadoras: emisor y receptor, a través de las pruebas de integridad y autenticidad.

El presente trabajo se centra en la vulneración del servicio de **confidencialidad**.

1.3.2. SERVICIO DE CONFIDENCIALIDAD

La información transmitida está dirigida a persona/s o equipo/s especificados por el emisor; sin embargo, en el canal a través del cual se transmite dicha información se considera inseguro y terceras personas pueden interceptarla. Una manera de garantizar el servicio de confidencialidad, de modo que solamente el o los receptores deseados puedan acceder a la información transmitida, consiste en cifrar la información (figura 1.3). De esta manera, si alguien más consigue aquella información, se encontrará con un conjunto de caracteres que a simple vista no tendrán sentido alguno [3].

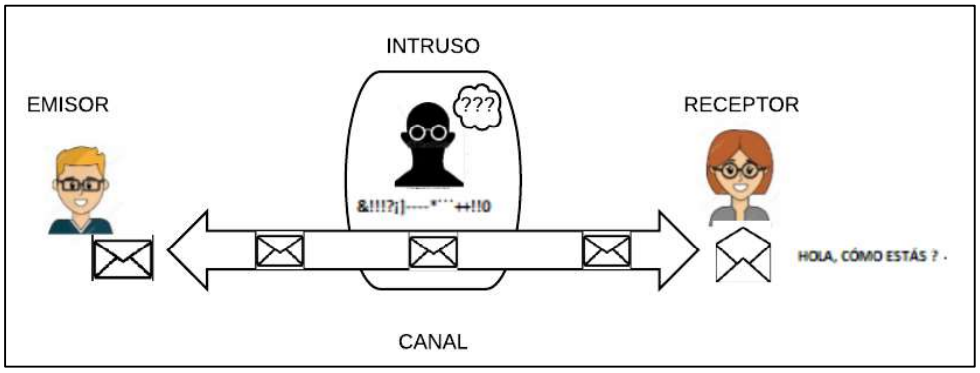


Figura 1.3. Envió y recepción de información en un canal de comunicación inseguro.

Este servicio de confidencialidad puede ser implementado por un algoritmo de cifrado que puede ser de clave simétrica o de clave asimétrica, el presente trabajo se centra en el primer caso.

Para que un algoritmo se considere de clave simétrica es necesario que el mensaje haya sido cifrado con una clave que deberá ser conocida por el o los receptores, de modo que se pueda descifrar correctamente al llegar a su destino. En este trabajo se intentará descubrir el texto original sin disponer de la clave secreta y se lo hará únicamente probando con cada una de las posibles claves y se analizará las características del texto descifrado para saber si fue o no la clave correcta.

1.3.3. DES (DATA ENCRYPTION STANDARD)

Este es un algoritmo de clave simétrica desarrollado por petición del gobierno de los Estados Unidos para suplir la necesidad existente de seguridad en las comunicaciones. Creado por la NSA(Agencia de Seguridad Nacional) en 1976, fue escogido como un estándar FIPS(Estándares Federales de Procesamiento de la Información). Es una modificación de la propuesta de IBM, el algoritmo conocido como LUCIFER (1974) [4].

Este sistema de cifrado utiliza bloques de texto de longitud fija igual a 64 bits, y una clave de igual valor, donde 8 bits son considerados de paridad. Por lo anterior solo son procesados 56 bits para la clave secreta (Figura 1.4).

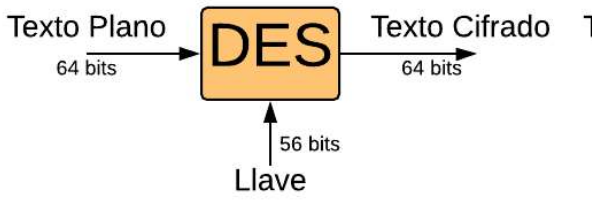


Figura 1.4. Entrada y Salida de Cifrado DES

El proceso de cifrado consiste en un conjunto de permutaciones y combinaciones entre el texto en claro y la clave de cada ronda, existen 16 rondas utilizando el esquema FEISTEL (Método de cifrado reversible), donde es necesario segmentar el texto a la mitad, para procesar cada sub segmento, el primero se intercambia con el segundo después de haber pasado por una operación XOR con el resultado de la Función F, para finalmente concatenar resultados de la ronda final y estos pasar a la permutación final, obteniendo un texto del mismo tamaño de ingreso pero cifrado (Figura 1.5).

El proceso de descifrado es similar al cifrado, debido al uso de FEISTEL, que como se mencionó anteriormente es reversible, lo cual permite que para descifrar el proceso sea igual; sin embargo, el orden en que se aplican las claves será invertido, empezando por K16 y finalizando por K1.

El algoritmo expuesto se consideró como el mejor para la seguridad no solo en Estados Unidos sino alrededor del mundo debido a su nivel de seguridad alto a pesar de tener una clave relativamente pequeña, esto hasta 1997 donde este fue considerado inseguro debido a que se logró romper en aproximadamente 3 meses.

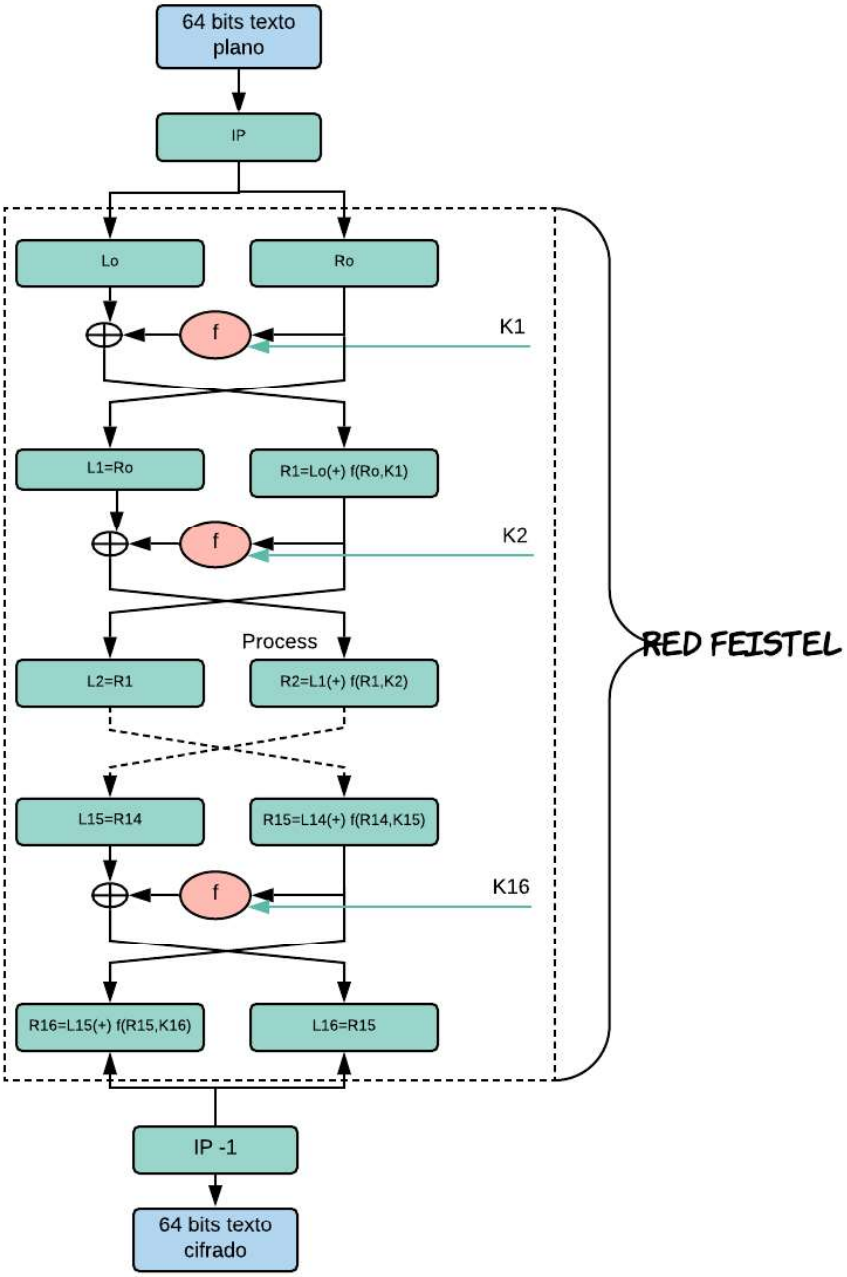


Figura 1.5. Diagrama de Flujo del algoritmo DES.

1.3.4. DES CHALLENGES

RAS Security Company en 1997 propuso un concurso abierto al público conocido como DES Challenge, cuyo objetivo era descifrar mensajes desconocidos que había sido cifrados con DES, para desprestigiar al Data Encryption Estándar ante el gobierno de los Estados Unidos, diciendo que este era ineficiente e inseguro. El premio era una gran cantidad de dinero, el proceso responde a un ataque por fuerza bruta, donde en este caso fue necesario probar 72 cuatrillones de claves.

No demoró el primer logro, después de la publicación inicial de un texto cifrado con DES, fueron Rocke Verser, Matt Curtin y Justin Dolske que constituyeron el proyecto DESCHALL utilizando computación distribuida a través de internet, su estructura estaba constituida por un servidor de claves administrado por uno de ellos y varios cliente, donde se ejecutaba un pequeño programa el cual permitía el procesamiento de ciertas claves en los ciclos de inactividad de las máquinas, encontrando el mensaje original en 96 días [7].

En la versión 2 del concurso, se mejoró el tiempo en que fue encontrado el texto original a 39 días, esto a cargo de Distibuted Computing Technologies, Inc, cuyo mensaje descifrado fue “Many hands make light work” [8]. Una mejor opción fue la propuesta por Electronic Frontier Foundation (EFF) donde solo fueron necesarias 56 horas para encontrar la clave y descifrar, supliendo la computación distribuida por una máquina creada únicamente con el propósito de probar una a una las posibilidades existentes en el espacio de claves conocida como DEEP CRAK [9].

Para la última versión del concurso, se unieron Electronic Frontier Foundation (EFF) y Distributed.Net y Electronic Frontier Foundation (EFF) logrando romper DES en menos de 1 día, aproximadamente 22 horas [10].

La Tabla 1.1. muestra un resumen de los desafíos DES en cada una de las versiones del concurso, con sus respectivos ganadores.

Tabla 1.1. DES Challenges

Versión del concurso	Fecha	Autores	Tiempo
DES Challenge I	junio de 1997	Rocke Verser, Matt Curtin y Justin Dolske	96 días
DES Challenge II-1	febrero de 1998	Distributed Computing Technologies, Inc	39 días
DES Challenge II-2	Julio de 1998	Electronic Frontier Foundation (EFF)	56 horas
DES Challenge III	Enero de 1999	Distributed.Net y Electronic Frontier Foundation (EFF)	22 horas y 15 minutos

1.3.5. S-DES (SIMPLIFIED DATA ENCRIPTION STANDARD)

S-DES es una adaptación sencilla del Data Encription Standard (DES) (Figura 1.6.) cuyo objetivo es facilitar su estudio, creado por Edward F. Schaefer en 1996, mantiene los parámetros originales del DES pero reducidos totalmente, de tal manera que sea comprensible. Los parámetros modificados se muestran en la tabla 1.2 donde se puede observar la diferencia entre los algoritmos en mención [5].

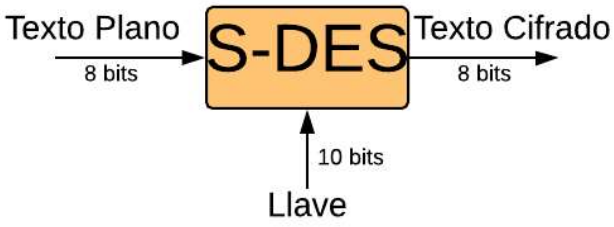


Figura 1.6. Envió, Recepción en Intercepción en un canal de comunicación

Tabla 1.2. Parámetros DES vs S-DES

Característica	DES	S-DES
Longitud de la clave	64 bits -56 bits utilizados	10 bits-10 bits utilizados
Longitud del texto de Entrada/Salida	64 bits	8 bits
Número de rondas Feistel	16	2
Longitud de las Sub-claves	48 bits	8 bits

Para el proceso de Cifrado y Descifrado, se realiza un conjunto de pasos constituidos por permutaciones, sustituciones y otras operaciones, a partir del texto en claro o el cifrado según sea el caso. Como se muestra en la figura 1.7, el proceso se exactamente igual al DES.

Para el cifrado, el texto pasa por la permutación inicial IP luego pasa a la función Feistel (fk) que requiere la generación de la sub-clave k1, obtenida después de un proceso de permutación, reducción y desplazamiento de un bit. A continuación, se utiliza por segunda ocasión fk, con la sub-clave k2 y finalmente se aplica la permutación que obtiene el texto cifrado. El mismo proceso es necesario para descifrar el texto cifrado, siempre y cuando se

conozca la clave con la que fue cifrado dicho texto, este beneficio ocurre por la utilización de la red Feistel que es reversible [6].

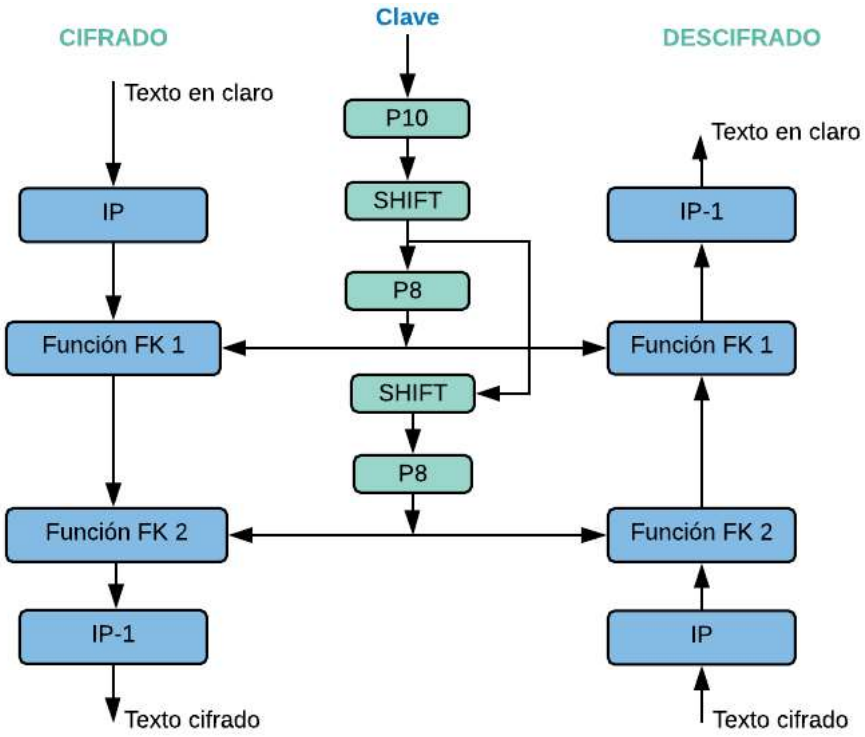


Figura 1.7. Cifrado y Descifrado S-DES

Si bien el algoritmo DES era considerado seguro, históricamente la empresa RSA llamó a distintos concursos para probar el grado de vulnerabilidad ante un ataque por fuerza bruta, esto lo se resume continuación:

Finalmente, el gobierno de los Estados Unidos declaró a DES como no confiable, ya que los datos estarían seguros por un día, forzando al fortalecimiento del mismo con la creación del Triple-DES y AES, el cual hasta el día de hoy no se ha podido romper.

1.3.6. ATAQUES CRIPTOGRÁFICOS

Según el estudio de la historia del Algoritmo DES, se puede decir que con el avance de la tecnología y la mejora en hardware y software la mayoría de algoritmos pueden ser vulnerados. Por ende, la información que protegían puede ser accedida por una persona externa, esto se ha conseguido con los Ataques criptográficos.

Un ataque criptográfico constituye una de las principales amenazas que existen dentro de las tecnologías de la información; consiste en aprovechar alguna debilidad o vulnerabilidad en el software, hardware e inclusive de las personas que tienen acceso a una red con el objetivo de obtener, alterar o suprimir información sin autorización, para un fin económico generalmente, esto provocando alteraciones en los recursos de una persona u organización. A continuación, se presentan los ataques más importantes que se pueden realizar a un algoritmo de cifrado simétrico por bloques (como lo son DES y S-DES).

a. Criptoanálisis Diferencial

Este ataque se basa en el análisis de un par te textos en claro que han sido cifrados con la misma clave, se realiza un análisis exhaustivo de datos con el objetivo de obtener la clave correcta, para esto se asignan probabilidades a cada clave posible. La cantidad de pares se incrementa hasta obtener la clave correcta [11].

b. Criptoanálisis Lineal

Si bien trabaja con pares de textos plano-cifrado como el criptoanálisis diferencial, el contraste radica en que solo se evalúa un número específico de bits dentro de los textos, de tal manera que se pueda realizar operaciones XOR entre ellos. Primero se seleccionan n bits en el texto plano y se realizan operaciones XOR entre estos, luego se seleccionan n bits en el texto cifrado y se aplica el mismo proceso anterior, obteniendo un bit y un bit para cada caso, entre estos nuevamente se utiliza XOR, este proceso se repite n veces, el objetivo es encontrar la mayor cantidad de 0s o 1s para asignar probabilidades más altas a cierta claves con el objetivo de encontrar la correcta [12].

c. Ataques Algebraicos

El éxito de este tipo de ataque está en analizar la estructura matemática que conforma en sí el algoritmo, con el objetivo de plantear ecuaciones con el número de incógnitas igual a la longitud de la clave y resolverlo [13].

d. Ataque por fuerza bruta o Búsqueda exhaustiva

Como se ha expuesto hasta el momento, la parte más importante de un algoritmo simétrico radica en el secreto de su clave, de modo que, si se la obtiene, se ha vulnerado completamente el algoritmo y por tanto se tiene acceso a la información escondida.

El ataque por fuerza bruta consiste en obtener la clave con la que un texto fue cifrado, sin necesidad de criptoanalizar el algoritmo. Por esta razón, la seguridad del algoritmo radica en la longitud de su clave, necesitando un mayor poder computacional mientras mayor sea

su tamaño. Este tipo de ataque se considera más sencillo que los otros tipos de ataques mencionados anteriormente; sin embargo, se requiere de altas prestaciones del equipo o los equipos que se utilicen. Entonces se probará una a una las posibilidades existentes; por ejemplo como se mencionó anteriormente DES tiene una clave de longitud neta de 56 bits, de modo que las posibilidades a evaluar serán 2^{56}.

e. Ataque por fuerza bruta por diccionario de palabras

En este caso, en lugar de ir probando cada una de las posibles claves, se prueba palabras de uso cotidiano hasta encontrar la clave correcta.

Este tipo de ataque es muy conocido dentro de internet, ya que se pretende encontrar las contraseñas para ingresar a diferentes páginas que exija una cuenta, al igual que para acceder a una red WI-FI. Existen varios programas que permiten este tipo de vulneraciones, de hecho, son de mayor facilidad ya que las contraseñas suelen ser de tamaño reducido.

Con el incremento de las prestaciones computacionales de los equipos de hoy en día, CPU con varios multiprocesadores, se puede realizar este tipo de ataque con mayor facilidad; sin embargo, si la longitud de la clave incrementa, un solo equipo no será capaz de lograrlo por lo que se hace necesario utilizar computación paralela.

En el presente proyecto se empleará el ataque por fuerza bruta o búsqueda exhaustiva ya que en comparación con los demás métodos presenta una menor dificultad respecto al análisis del texto, únicamente se centra en las posibilidades de la clave.

1.3.7. COMPUTACIÓN PARALELA

Los programas o aplicaciones tradicionalmente son creados para una ejecución secuencial, se dividen en instrucciones que se dirigen al CPU para ser procesadas una por una según su orden de llegada (Figura 1.8).

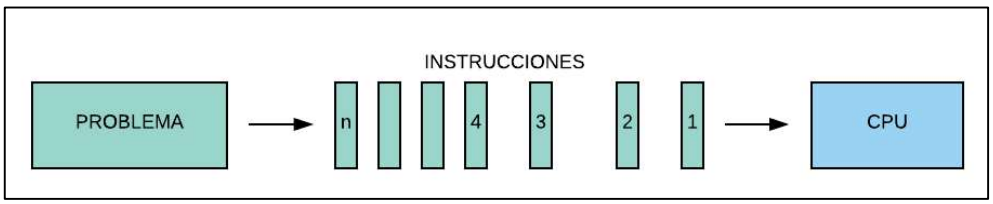


Figura 1.8. Procesamiento Secuencial.

La Computación paralela consiste en el uso de múltiples procesadores o computadoras, las cuales trabajan juntas y simultáneamente para resolver un problema o solventar una tarea común [14]. El total de las instrucciones independientes en que se haya dividido el problema es distribuido entre varios procesadores que pueden estar dentro de un mismo CPU o en distintos CPU en el caso de clúster (Figura 1.9).

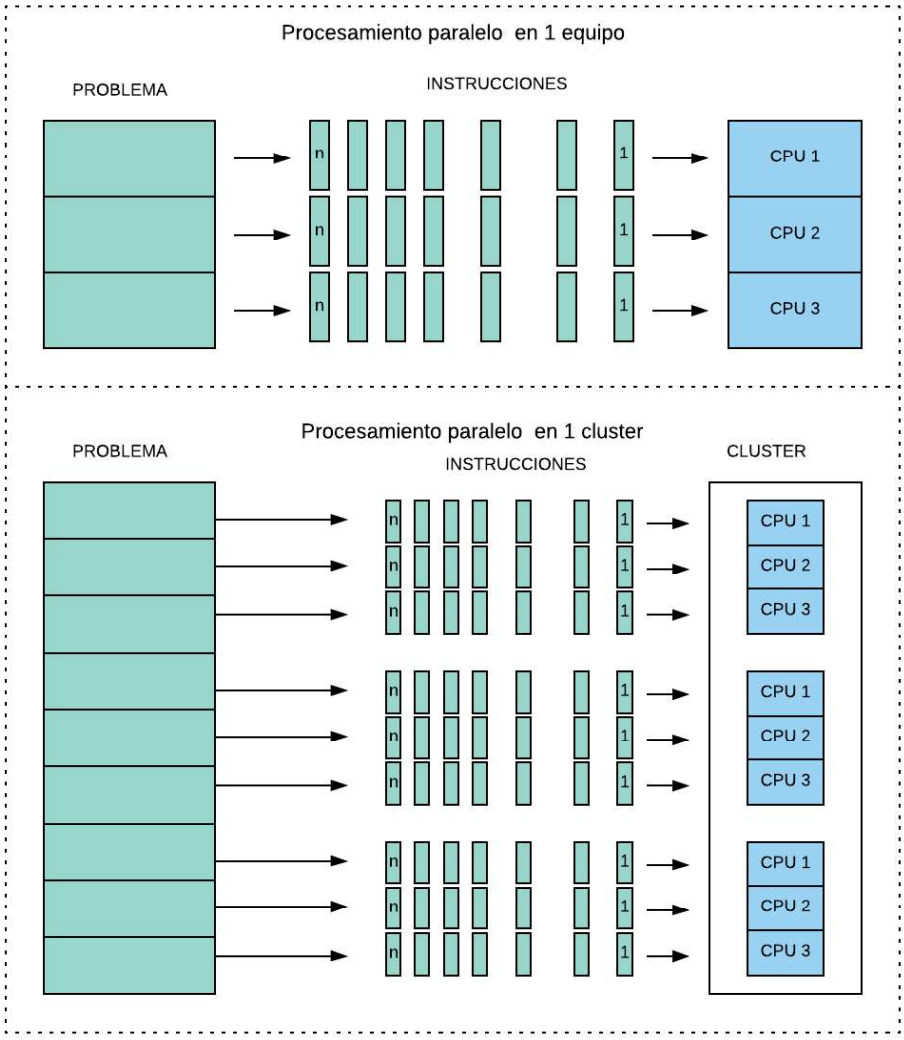


Figura 1.9. Procesamiento Paralelo.

La computación paralela es utilizada por las limitaciones que existen en la secuencial, siendo los motivos principales:

- Resolución de problemas en menor tiempo: la disminución de tiempo se presenta por la concurrencia de operaciones en todos los procesadores.

- Resolución de problemas complejos: se pueden resolver problemas de gran complejidad ya que la ejecución se la realiza con múltiples procesadores.
- Procesamiento de gran cantidad de datos: se pueden procesar conjuntos de datos grandes y distribuirlos entre procesadores.

1.3.8. ORIGEN DE LA COMPUTACIÓN PARALELA

En los años 50 ya se hablaba del paralelismo; sin embargo, las ideas aun no eran plasmadas físicamente, en 1956 Stephe Viger propuso un conjunto de procesadores organizados como un arreglo con memoria, en 1958 J. Holland planteó la ejecución de varios programas independientes, simultáneamente. En este mismo año S.Gill Ferranti estudió la computación paralela y la necesidad de dividir en partes un proceso [15].

En los años 60 todo lo teórico se ve plasmado en una máquina paralela denominada Solman, creada por Westinghouse Electric [16]; sin embargo, este tipo tecnología generaba grandes gastos. En los años 70 aparecen las super computadoras CRAY las mismas utilizaban un procesamiento por vectores, en los años 80 con el uso de microprocesadores surgen computadoras CRAY modificadas de tal manera que pueden tener de 1 hasta 16 procesadores, el tiempo de procesamiento era relativamente pequeño; sin embargo, el tiempo en espera en la cola era grande. En este mismo año se originan las estaciones de trabajo que eran solamente 2 veces más lentas que las CRAY.

En la década de los 90 la computación paralela se extendió con la aparición de sistemas como INTEL, Thinking Machines, nCUBE.

En la actualidad se tiene la posibilidad de utilizar el paralelismo, en supercomputadoras, procesadores multinúcleo, redes de computadoras, sistemas distribuidos, tarjetas gráficas y de video e incluso en una PC de uso común [17].

1.3.9. MODOS DE PARALELISMO

Se ha clasificado al paralelismo de acuerdo al esfuerzo que realiza el programador al elaborar el algoritmo o la máquina que ejecuta el programa.

a. Paralelismo implícito

Este trabajo lo realiza el compilador puesto que el algoritmo estará escrito de manera secuencial, una vez en ejecución será analizado y organizado, incluso insertando ciertas instrucciones necesarias con el objetivo de convertirlo en un algoritmo más eficiente que el serial, en este caso el esfuerzo invertido por el programador es pequeño.

b. Paralelismo explícito

El usuario deberá detallar como distribuirá las tareas, procesos o funciones que se realizan dentro del algoritmo, entre los procesadores, siendo así el programador el que realiza su mayor esfuerzo.

En la presente aplicación se utiliza el paralelismo implícito dado que no se tiene el control de la distribución de tareas a cada procesador; sin embargo, el código se modifica utilizando comandos de herramientas paralelas.

1.3.10. NIVELES DE PARALELISMO

a. Paralelismo a nivel de bit

Depende de la cantidad de información que se pueda manejar por ciclo del procesador.

b. Paralelismo a nivel de instrucción

Dado que un programa computacional está formando por un conjunto de instrucciones, este tipo de paralelismo consiste en reordenar y combinar dichas instrucciones para que sean ejecutadas sin que el resultado se afecte.

c. Paralelismo a nivel de tarea

Se asignan 1 o más tareas a cada procesador.

d. Paralelismo a nivel de datos

Todos los procesadores manejan diferentes datos, pero realizan las mismas acciones.

La presente aplicación realiza un paralelismo a nivel de datos: si se considera una matriz de claves, estos valores se segmentan y cada procesador evalúa un conjunto de datos diferente simultáneamente.

1.3.11. PARADIGMAS DE LA PROGRAMACIÓN PARALELA

a. Memoria compartida

La comunicación entre procesadores se da accediendo a una memoria común donde estarán almacenadas variables u objetos necesarios para la ejecución de un programa, evita las copias redundantes en cada uno y puede ser en un solo procesador o muchos. Puede ser un modo eficaz de comunicación; sin embargo, se tiene el problema de acceso de varios procesadores al mismo elemento de la memoria, para lo cual se debe manejar un protocolo de semáforo, o utilizar el paso de mensajes que se explica a continuación.

b. Paso de mensajes

Los procesadores no comparten memoria, variables u objetos, si no que para comunicarse utilizan operaciones de envío y recepción. Esto soluciona el problema que existe en los sistemas de memoria compartida. La comunicación puede ser asíncrona, es decir, se puede enviar un mensaje al receptor, aunque no esté listo para recibirlo o síncronas donde el receptor debe estar listo.

La ejecución paralela del presente algoritmo no utiliza memoria compartida, los trabajadores creados en cada procesador se comunican durante la ejecución.

1.3.12. CLÚSTER COMPUTING

Los altos costos que generan el uso de supercomputadoras han dado lugar a la utilización de clústeres, los cuales representan una forma económica de acceder a una potencia informática alta para resolver problemas [18].

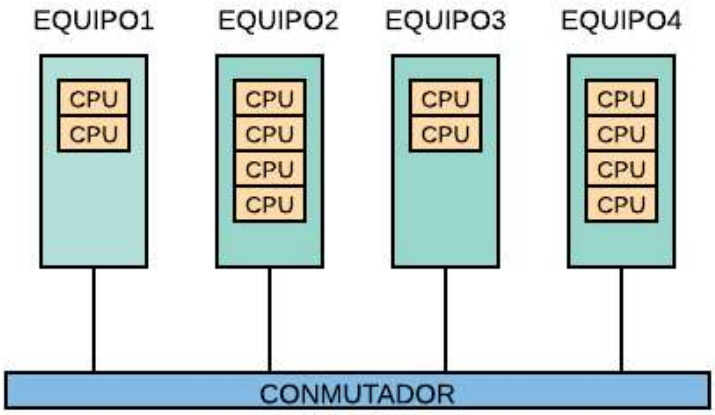


Figura 1.10. Clúster de 4 nodos.

Un clúster es un conjunto de computadoras paralelas o distribuidas, conectadas entre sí mediante redes de alta velocidad (figura 1.10), trabajan juntas con el propósito de resolver tareas que conllevan grandes recursos computacionales o que incluyan grandes cantidades de datos, trabajo que un solo computador de uso general no podría realizarlo por sí mismo. Sin embargo, no solo es trabajo de hardware, sino que se necesita un software que permita la conformación del grupo y su administración, además admita la interacción con el usuario.

Si un nodo falta, los demás realizarán la ejecución con una distribución de recursos diferente; sin embargo, no habrá un error [19]. Para el incremento de la potencia de

procesamiento, únicamente es necesario añadir un nodo (equipo) adicional. Los nodos pueden ser dedicados y no dedicados [20].

Dedicados: su uso está exclusivamente dedicado a tareas relacionadas con el clúster, no dispone de periféricos conectados como: mouse, monitor o teclado, etc.

No dedicados: Todos los nodos tienen periféricos de entrada y salida, por ende, puede realizar tareas externas al grupo de computadores, por lo cual el clúster utiliza los ciclos de reloj disponibles.

La presente aplicación se ejecuta en un clúster conformado por nodos no dedicados, 3 laptops.

Los clústeres pueden ser homogéneos o heterogéneos (Figura 1.11) los primeros consisten en conjuntos de computadoras que tienen una misma arquitectura y recursos similares, por ende, los nodos son técnicamente iguales. En los segundos, los nodos pueden tener diferente arquitectura, procesadores, sistemas operativos, etc.

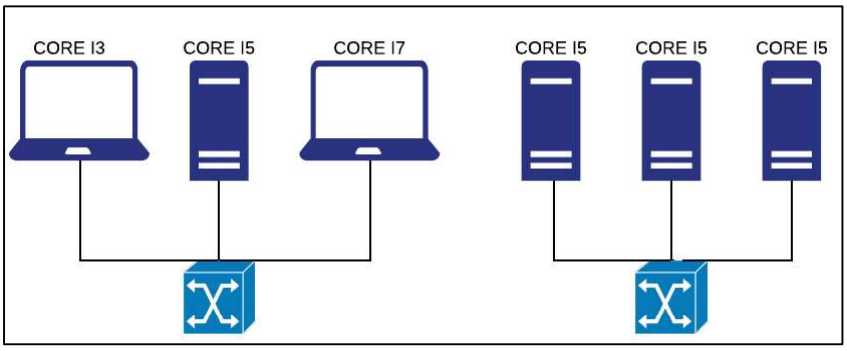


Figura 1.11. Clúster heterogéneo(izquierda) y clúster homogéneo(derecha)

Físicamente un clúster es un conjunto de equipos donde las tareas son divididas entre sus procesadores; sin embargo, para la vista del usuario, trabaja como una única computadora.

1.3.13. COMPONENTES DE UN CLÚSTER

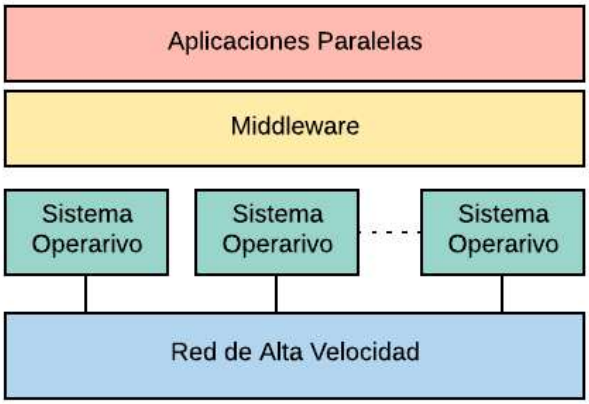


Figura 1.12. Arquitectura de un Clúster

La arquitectura de un clúster (Figura 1.12), está formada por:

a. **Aplicaciones Paralelas:** son aplicaciones desarrolladas dentro de un ambiente de programación paralela que dan solución a un problema específico.
b. **Middleware:** es un software que coordina los trabajos en una aplicación desarrollada dentro de un clúster, distribuye los mensajes o tareas a través de la red entre los nodos que conforman la agrupación, la distribución evita sobrecargas en un nodo optimizando el funcionamiento, de modo que, si un nodo está demasiado cargado con un proceso y otro sin uso, se puede liberar carga, de esta manera se tendrá rapidez en la ejecución. La redistribución de trabajo se puede establecer manual o automáticamente [21].
c. **Sistema Operativo:** software que permite la administración de recursos de Hardware y software dentro de un computador.
d. **Red de alta velocidad:** conjunto de dispositivos que interconectan varios equipos a velocidades superiores a 10 Mbps, utilizando diferentes tipos de tecnologías, como Ethernet y sus variantes, ATM, Myrinet, etc. Un clúster no se conecta mediante redes exclusivamente de “alta velocidad”.

Existen componentes secundarios que son parte de los que se han especificado anteriormente, específicamente dentro de los nodos:

e. Elementos de Hardware

Los nodos pueden ser de diferentes marcas y por supuesto tener diferentes prestaciones y de esto dependerá su actuación dentro de la ejecución de un algoritmo paralelo.

El procesador comprende la parte central de una computadora, se encarga del total control de las tareas y programas que se ejecutan en el equipo, puede contener más de una Unidad Central de Procesamiento (CPU) que es lo común en tiempo actuales, esto llamado multiprocesador multinúcleo.

La memoria Cache trabaja con CPU, es una memoria temporal que acelera el tiempo que toma en realizar consultas a la memoria principal, la primera vez que se accede algún lugar específico realiza una copia de datos desde la memora principal para tenerlos disponibles casi de inmediato la próxima vez que se intente acceder a este mismo lugar, es relativamente pequeña respecto a su tamaño.

La memoria RAM alberga las utilidades y datos con los que trabaja en un determinado momento, si es pequeña habrá limitaciones lo que causa la percepción de lentitud en el equipo.

La memoria ROM es permanente y contiene programas que permiten el arranque del equipo.

1.3.14. TIPOS DE CLÚSTER

a. Alto rendimiento (High Performance Computing):

Este tipo de clúster está destinado para ejecutar tareas que requieren de gran capacidad de cálculo matemático, renderización de gráficos, compilación de programas, manejo de grandes cantidades de datos, descifrado de códigos [22].

b. Alta eficiencia (High Throughput Computing)

Su objetivo es ejecutar la mayor cantidad de tareas en el menor tiempo posible. Para esta clasificación es mucho más eficiente el uso del GPU (Graphics Processing Unit).

c. Alta disponibilidad (High Availability Computing)

Su propósito es garantizar la máxima disponibilidad y estabilidad de los servicios que ofrecen. Para esto es necesario la utilización de recursos de forma redundante. Utilizados en su mayoría por empresas [23].

El clúster utilizado en este caso es de alta eficiencia, dado que el objetivo es conseguir el menor tiempo posible en el procesamiento de claves.

1.3.15. HERRAMIENTAS ACTUALES

Como se sabe, existe un sinnúmero de lenguajes de programación que actualmente permiten el uso del paralelismo mediante librerías, directivas o toolboxes que deberán ser instaladas para que se pueda programar códigos cuya funcionalidad será paralela. A continuación, se mencionan los lenguajes de programación más conocidos que permiten desarrollar esta funcionalidad:

a. C++

Lenguaje extensión de C, considerado de nivel medio, permite la programación estructurada y orientada objetos, considerado por tal motivo como lenguaje híbrido. Permite la utilización de programación paralela mediante del uso de librerías conocidas como OpenMP y MPI (Message Passing Interface).

Open Multi-Processing (OpenMP)

Estándar de memoria compartida. Permite el manejo simultaneo de hilos (subprocesos) dentro de un mismo procesador. Así que al inicio se pone en marcha 1 hilo, este puede dar origen a varios dependiendo de la tarea a realizarse [24].

Message Passing Interface (MPI)

Se basa en la cooperación de varios procesadores para resolver un problema, sin embargo, estos no comparten memoria, por lo cual es necesario el paso de mensajes para que los procesos se comuniquen.

b. FORTRAN(*Basic Linear Algebra Subprograms*)

Es el primer lenguaje de alto nivel, está orientado al procedimiento, es decir que el programador escribe las instrucciones con un orden específico para resolver un problema, es utilizado para el desarrollo de aplicaciones científicas y el análisis numérico [25]. Durante su evolución ha pasado de ser un lenguaje imperativo a uno orientado a objetos. En lo que respecta a la computación de alto rendimiento es bastante popular. Para la implementación de la computación paralela maneja 2 librerías al igual que C++, OpenMP y MPI.

c. JAVA

JAVA es otro derivado de C, por lo cual maneja una sintaxis similar, es un lenguaje de programación orientado a objetos, su principal característica es que lo programado aquí,

puede ser ejecutado en cualquier lado sin importar la arquitectura de la computadora, esto conocido como independencia de plataforma, esto gracias a la JAVA Virtual Machine (JVM); Las aplicaciones se compilan dando como resultado bytecode el cual es propio de JAVA, esto se ejecutará sobre cualquier dispositivo que tenga previamente instalado la JVM. Para poder correr aplicaciones paralelas utiliza *Java Parallel Processing Framework* es un framework y MPI.

Java Parallel Processing Framework

Permite ejecutar aplicaciones que requieren gran cantidad de potencia de procesamiento en varios equipos, con el objetivo de reducir el tiempo de procesamiento, lo que hace es dividir la aplicación en pequeñas partes para que sean ejecutadas simultáneamente en los diferentes procesadores del grupo de máquinas, se utiliza en Grid Computing. [26].

d. MATRIX LABORATORY (MATLAB)

Matlab es un software matemático que maneja la programación en lenguaje m, el elemento básico es una matriz que no necesita ser dimensionada y está basado en las bibliotecas de algebra: LAPACK(*Linear Algebra Packege*) y BLAS(*Basic Linear Algebra Subprograms*) [27], además de manejar un ambiente de programación brinda el manejo de gráficos y simulaciones.

Presenta un conjunto de cajas de herramientas utilizadas para resolver problemas particulares, uno de ellos es la programación paralela que utiliza: *Parallel Computing Toolbox y MATLAB Distributed Computing Server.*

Matlab permite, mediante el uso del PCT (*Parallel Computing Toolbox*), utilizar múltiples *workers* en una sola computadora y con ayuda del DCS (*Distributed Computing Server*) múltiples *workers* en varias computadoras formando un clúster (figura 1.13)

Parallel Computing Toolbox

Permite resolver problemas de alto costo computacional utilizando procesadores multinúcleo, GPU y clústeres, contiene un conjunto de comandos que permiten paralelizar ciertos segmentos de código secuencial, como bucles y matrices distribuidas. Permite la programación paralela sin MPI. Tiene un conjunto de comandos que permiten modificar la programación secuencial, se analizará los principales:

1. Parallel for-loop (PARFOR)
2. Single Program Multiple Data (SPMD)

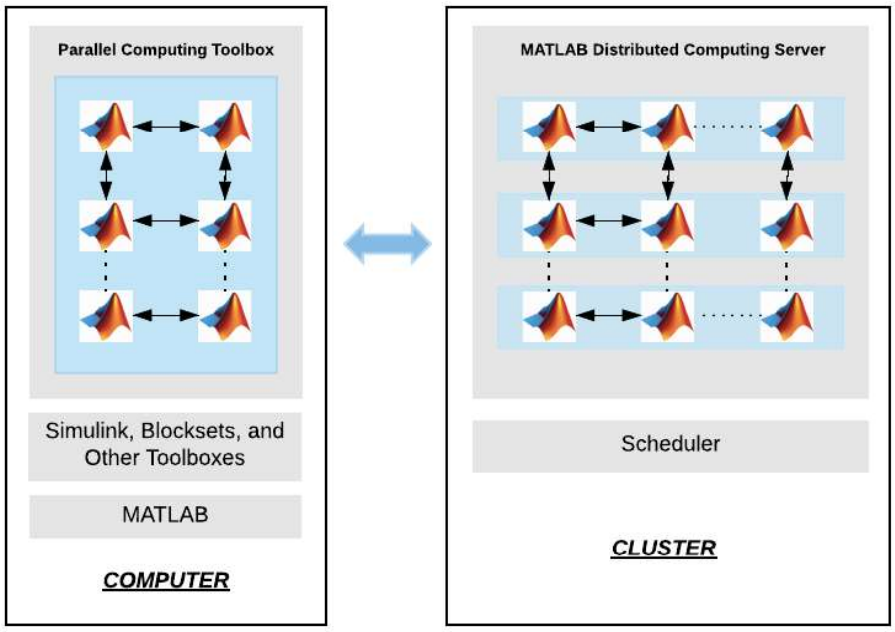


Figura 1.13. Programación paralela en Matlab (en la izquierda Paralelo Local y en la derecha paralela en clúster).

- ✓ ***Parallel for-loop*** **(Parfor)**

Gracias a la herramienta *Paralell toolbox s*e puede cambiar ciertas funciones por otras para paralelizar, como principal ayuda aparece el PARFOR, que remplaza todo ciclo repetitivo siempre y cuando cumpla con las condiciones que establece, con su uso automáticamente, el entorno mandará una orden interna a la CPU para que reparta el cálculo que está realizando entre el total de sus núcleos. La carga de trabajo se distribuye de manera uniforme y automática de acuerdo con el índice de bucle, Los detalles son transparentes con el usuario.

Si el proceso dentro de un bucle no es suficientemente grande o complejo, no sería útil la distribución entre procesadores ya que, en lugar de reducir el tiempo, lo incrementará cuando todos los *workers* se comuniquen.

Para que el bucle de parfor funcione, las variables dentro del bucle deben ser parte de las categorías de la tabla 1.3.

Tabla 1.1. Tipos de variables dentro de un PARFOR.

Categoría de variable	Descripción
Loop	Variable utilizada como índice del bucle para arrays
Sliced	Variable cuyos segmentos son utilizados en cada iteración, pueden ser de entrada y salida.
Broadcast	Es una variable definida previo al bucle, este valor se utiliza dentro del mismo, pero no es asignado adentro.
Reduction	Es una variable que incrementa su valor conforme pasan las iteraciones del bucle, independiente del orden en que se las realiza.
Temporary	Es una variable creada dentro del bucle, pero, difiere de la variable del sliced, dado que no estará disponible fuera del bucle.

```
t=0;
b=pi;
r=0;
si=rand (1,10);
parfor i=1:10
    t=i;  ------------------>  t: variable temporal- i: variable
loop
    r=r+i;  ----------------> r: variable reduction
    so(i)=si(i);  ----------> so: salida sliced- si: entrada sliced

    if i<=b  ---------------->  b: variable broadcast
        resul=2*t;
    end
end
```

Segmento de código 1.1. Detalle de tipos de variables PARFOR.

Un bucle parfor generará un error si contiene cualquier variable que no puede ser categorizada en una clase de variable o si las variables violan sus restricciones de categoría al ser clasificadas.

Problemas y soluciones

Como se expuso anteriormente parfor no siempre es posible, además de la clasificación de variables, otro problema importante es la existencia de recursividad, de modo que el bucle no siempre puede obtener el valor de la iteración anterior ya que estas no se dan en

un orden específico y el programador tampoco podrá establecer una disposición. El siguiente ejemplo permite generar y visualizar mediante consola los números del 1 al 5, lo que se esperaría en la salida si fuera un ciclo for sería “1-2-3-4-5”, pero la salida es la que se muestra en la figura 1.14.

EJEMPLO

```
parfor i=1:5
disp(i);
end
```

Segmento de código 1.2. Ejemplo simple de PARFOR.

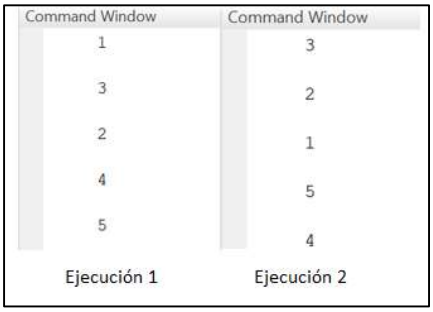


Figura 1.14. Salida ejemplo simple de PARFOR.

Ejecutando nuevamente este código, se obtiene una salida diferente, lo que permite concluir que la recursividad no es posible realizar dentro de un ciclo parfor.

Existen otros problemas dentro de bucle parfor, que Matlab advierte con un error, sin embargo, estos si tienen una solución, a continuación, se presentan cada uno con un ejemplo para un mejor entendimiento.

- ✓ ***Contador no entero***

Cuando el valor de incremento de cada iteración no es un número entero, Matlab da como resultado un error, el siguiente código representa un bucle que se espera vaya en incremento de 0.2, partiendo de 0 hasta llegar a 1.

EJEMPLO:

```
parfor
i=0:0.2:1
disp(i);
      end
```

Segmento de código 1.3. Problema de contador no entero.

Al ejecutarlo, el mensaje que se obtiene es: parfor_range_check, como lo indica la figura 1.15.

```
Error using parfor_range_check (line 28)
The range of a parfor statement must be increasing consecutive integers.
```

Figura 1.15. Error PARFOR_RANGE_CHECK.

SOLUCIÓN:

El procedimiento para ejecutar y no obtener un error consta en utilizar una variable auxiliar "aux" en este caso, la cual será declarada fuera el bucle y cuyo contenido será la condición de interacciones, así que se forma un vector con los índices necesarios. Dentro de la declaración parfor, se pretende evaluar dicho vector, por lo cual el índice irá desde 1 hasta la longitud del mismo, para obtener los valores en cada vuelta, basta con ingresar al índice i del vector aux.

```
aux=0:0.2:1;
parfor
i=1:length(aux)
disp(aux(i));
end
```

Segmento de código 1.3. Solución de contador no entero.

El resultado por consola será el deseado, como se muestra en la Figura 1.16.

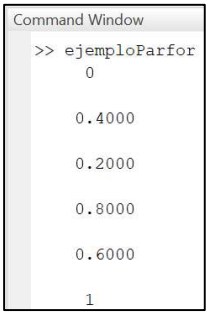


Figura 1.16. Salida en consola contador no entero.

- ✓ **Bucles anidados**

Cuando se intenta utilizar parfor dentro de otro parfor, existirá un error en la ejecución el cual no permite que el parallel pool inicie, por lo cual nunca se podrá completar la acción, como se observa en el ejemplo, cualquiera que sea el procedimiento que se encuentre formando parte del segundo ciclo paralelo no se podrá realizar, debido al error que se muestra en la figura 1.17, que menciona ParFor loops cannot be nested.

EJEMPLO:

```
parfor i=1:3
    parfor j=1:3
      %procedimiento
    end
end
```

Segmento de código 1.4. Problema de bucle anidado.

```
Error using ejemploParfor (line 2)
Error: Parfor loops cannot be nested.
```

Figura 1.17. Error PARFOR loops

SOLUCIÓN

Para poder ejecutar correctamente el script, paralelizando todo el contenido de los 2 ciclos parfor, se construye una función que contendrá como cuerpo el segundo parfor, de tal manera que esta será llamada desde el primer parfor sin ningún inconveniente en la ejecución.

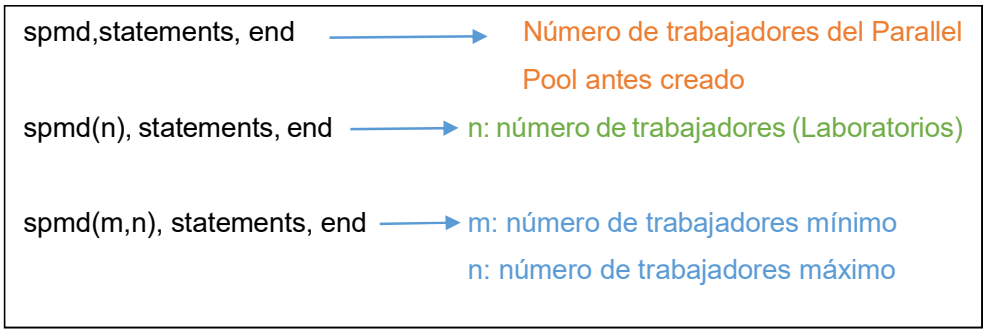
```
parfor i=1:3
   parfor2(i);
end

function parfor2(i)
 parfor j=1:3
```

Segmento de código 1.3.5. Solución de bucle anidado.

- ✓ **Single Program Multiple Data(SPMD)**

Otra herramienta alternativa para paralelizar el SPMD, cuyo nombre se descomponen para mejor comprensión; *Single Program*: se refiere a que el mismo código corre en diferentes trabajadores; *Multiple Data*: cada trabajador puede tener datos únicos y diferentes para el código. Los workers se conocen como laboratorios.

La sintaxis puede ser:



Segmento de código 1.6. Sintaxis SPMD.

Para ejecutar este comando, debe haberse creado un parallel pool obligatoriamente mediante código o a través de la interfaz gráfica esto detallado anteriormente.

Dentro del cuerpo de spmd existen valores únicos para cada laboratorio labindex, el número de laboratorios numlab indica cuantos trabajadores actuarán en la ejecución, los laboratorios tienen una comunicación entre ellos punto a punto, utilizan las funciones labsend, labrecive.

SPMD es eficiente cuando los programas tardan demasiado tiempo para ejecutar y cuando los programas trabajan con grandes cantidades de datos, permitiendo que varios laboratorios ejecuten soluciones simultáneamente y que los datos se distribuyan a varios laboratorios.

EJEMPLO:

Se presenta un ejemplo sencillo para comprender el funcionamiento del comando descrito, este conjunto de líneas de código tiene como objetivo utilizar 2 laboratorios, cada uno para realizar una parte del ciclo for, el trabajo es imprimir por consola números del 1 a 10, la condición indica que si el labindex (identifica al laboratorio) es 1 realice la impresión de los 5 primeros números de lo contrario muestre los últimos 5.

```
spmd
for i=1:10
  if labindex==1 & i<=5
 disp(i)
  else
  if labindex==2 & i>=5
          disp(i)
  end
  end
end
end
        %procedimiento
    end
   end
```

Segmento de código 1.7. Ejemplo de manejo de SPMD y condicionales.

La salida obtenida es la que se muestra en la figura 1.18, de modo que se puede elegir que hace cada laboratorio de acuerdo al funcionamiento del código a paralelizar.

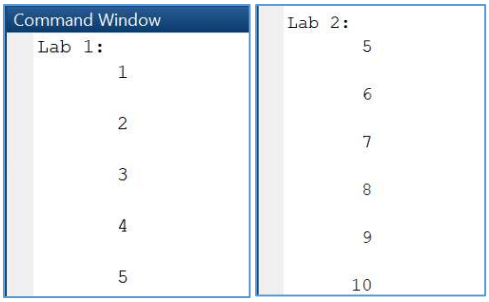


Figura 1.18. Salida en consola de manejo de SPMD y condicionales.

EJEMPLO:

Otra manera de manejar la distribución de trabajo entre laboratorios es SWITCH-CASE dentro del cuerpo SPMD, de manera que la variable que evaluará los casos será labindex, de acuerdo al número de identificación de laboratorio realizará la acción especificada dentro del caso (segmento de código 1.9), la salida será la misma que se muestra en la figura 1.18.

```
switch labindex
%Existen 2 casos, por ende 2
trabajadores
        %Trabajador 1
        case 1
           for i=1:5
           disp(i)
           end
        %Trabajador 2
        case 2
           for i=5:10
           disp(i)
           end
    end
```

Segmento de código 1.8. Uso de SWITCH-CASE.

EJEMPLO:

Se puede crear objetos compuestos, los cuales almacenan valores específicos para cada trabajador que se ejecuta dentro de SPMD, es similar a una matriz con celdas, donde cada elemento le pertenece a un laboratorio en ejecución, se maneja mediante índices dentro de llaves { }, de manera que se desea guardar valores para el laboratorio 2 se escribirá:

NombreObjeto{2}=valorAsignado;

A diferencia de otras variables en MATLAB, esta se construye, utilizando el constructor Composite () (segmento de código 1.10).

```
x=Composite();
%Cada índice indica el
labindex de cada
trabajador
%Se genera una matriz de
diferente dimensión para
que trabaje lab 1 y lab 2
x{1} = rand(5)
x{2} = rand(6)
```

Segmento de código 1.9. Uso de composite.

MATLAB Distributed Computing Server

Trabaja conjuntamente con *Parallel Computing Toolbox*, permitiendo ejecutar programas computacionalmente intensivos en conjuntos de computadoras denominados clúster o

grids, se admite trabajos por lotes, procesamiento de conjuntos grandes de datos entre otros, permite la instalación y ejecución del servicio MDCE para constatar la conexión entre todos los nodos que ejecutarán los programas paralelamente.

El servidor de computación distribuida de MATLAB le permite ejecutar un uso computacional intensivo programas MATLAB y modelos Simulink en agrupaciones de computadoras, nubes y cuadrículas, lo que le permite acelerar los cálculos y resolver grandes problemas.

La tabla 1.4. presenta un resumen de las herramientas necesarias para desarrollar y ejecutar un programa que ha sido paralelizado, dependiendo del software elegido.

Tabla 1.2. Librerías, directivas o *toolboxes* para algoritmos paralelos según el lenguaje de programación.

Lenguaje	**Librería, directiva o *toolbox***
C++	MPI Open MP
FORTRAN	MPI Open MP
JAVA	*Java Parallel Processing Framework*: MPI
MATLAB	*Parallel Computing Toolbox* *MATLAB Distribute Computing Server*

En este caso se utiliza MATLAB con sus herramientas *Parallel Computing Toolbox* y *MATLAB Distribute Computing Server* que permiten la construcción de un clúster y la paralelización del algoritmo con una ligera modificación en el código secuencial.

1.3.16. MEDIDAS DE RENDIMIENTO DEL PARALELISMO

Para evaluar un algoritmo secuencial se toma en cuenta únicamente el tiempo de ejecución de acuerdo a la complejidad del problema que resuelve; sin embargo, para evaluar un algoritmo paralelo, esto no es suficiente ya que no solo depende del tamaño del problema sino del número de procesadores que serán parte de esta ejecución.

A continuación, se especificarán medidas para evaluar el comportamiento de una arquitectura paralela como: tiempo de ejecución, seedUp, eficiencia.

a. Speed-up

En los ambientes paralelos es de interés poder ver que tan bueno es un algoritmo comparado con su respectivo serial en relación a la rapidez, así se podrá establecer si existe un beneficio con la utilización del paralelismo.

Speed-up es la aceleración que se obtiene al ejecutar un conjunto de líneas de código en un ambiente paralelo, en general, es la medida de la mejora del performance al aumentar procesadores. Es la relación que existe entre los tiempos de ejecución cuando se utiliza 1 procesador (serial) y cuando se utilizan n procesadores(paralelo). [28]

$$S = \frac{t_S}{t_n} \tag{2.1}$$

$$ts: tiempo\ en\ ejecutar\ el\ algoritmo\ secuencial\ (con\ 1\ procesador)$$

$$tn: tiempo\ de\ ejecución\ de\ algoritmo\ paralelo\ con\ n\ procesadores$$

Se puede establecer 2 tipos de Speed-up: absoluto y relativo, estos se diferencian por la definición del tiempo de ejecución secuencial ts.

a.1. Speed-up absoluto

El tiempo ts es el que se obtiene usando el mejor algoritmo secuencial para resolver el problema, en el caso de que no exista esta clasificación, utilizar para el cálculo el algoritmo serial más conocido.

a.2. Speed-up relativo

El tiempo ts es el que se obtiene con la ejecución el algoritmo en un solo procesador.

b. Eficiencia

La eficiencia de un algoritmo paralelo se define como el cociente entre la speed-up y el número de procesadores en utilización.

$$E = \frac{S}{N} \tag{2.2}$$

$$S: Speed\ Up\ y\ N: número\ de\ procesadores$$

c. Escalabilidad

Se dice que un sistema es escalable cuando la eficiencia se mantiene constante en valores superiores a 0.5 para un rango de procesadores.

En este caso se consideró los tiempos de ejecución para un texto corto y largo, con estos se realizó el cálculo de las speed-up y la eficiencia además la verificación de escalabilidad, en la parte de resultados se podrá apreciar el análisis.

1.3.17. DESARROLLO Y ESTRUCTURA DE UN ALGORITMO

Un algoritmo es un conjunto de instrucciones que realizan una tarea en específico, tiene un inicio y fin.

Para desarrollar un algoritmo independiente del ámbito para el cual se realice, es primordial el reconocimiento y análisis del problema que se pretende resolver, comprendido esto, se establecen las entradas que se necesitará para el desarrollo y las salidas esperadas.

Se puede utilizar ciertas herramientas como pseudocódigo, que consiste en escribir en cada línea una instrucción en palabras, especificando variables, valores y operaciones futuras, esto con el objetivo de tener una mayor facilitar al pasar esto a líneas de código en el software en el que se ha decidido programas.

Los diagramas de flujo es una representación gráfica de un algoritmo, donde se utilizan diversas formas que han sido estandarizadas por la ANSI e ISO, permite una organización clara y precisa de las instrucciones que se pretenden ejecutar en un futuro, en el cuadro de procesos se establecerán las operaciones utilizando los símbolos correspondiente, de la misma manera en el símbolo condicional que es un rombo, estos siendo algunas de las formas utilizadas en la elaboración de un diagrama de flujo, en la figura 1.19 se muestran sus elementos.

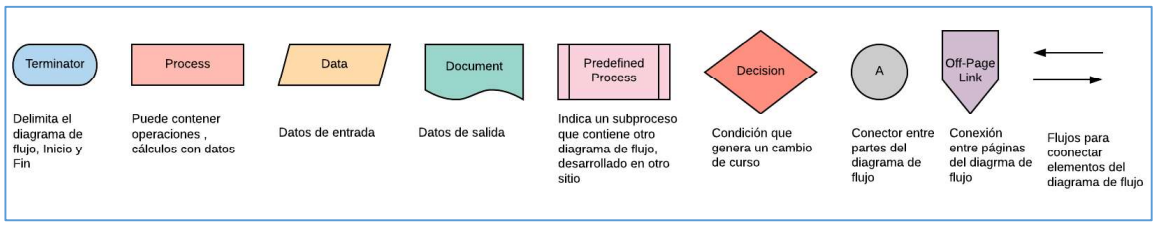


Figura 1.19. Elementos de un diagrama de flujo

Para comprobar la funcionalidad el algoritmo realizado mediante pseudocódigo o diagrama de flujo proceder a realizar una prueba de escritorio, esta consiste en probar si se satisface

todas las cláusulas descritas en el problema, dando diferentes valores a los argumentos de entrada.

Posterior a esto, se codificará de acuerdo al lenguaje elegido, originando varias sentencias que cumplan con lo propuesto.

a. Algoritmo paralelo

Un algoritmo paralelo es el que se puede dividir en partes para que estas sean ejecutadas por varios procesadores al mismo tiempo, de tal manera que al finalizar la ejecución se puedan unir resultados y obtener la solución del problema que evalúa dicho algoritmo.

Algunos algoritmos son fáciles de dividir, sin embargo, de acuerdo a la complejidad del problema se presentan inconvenientes para hacerlo, algo adicional es que, una instrucción no puede depender de otra ya que todas se realizarán al mismo tiempo por lo cual se introducen errores.

Cuando se ejecuta un algoritmo secuencial, las principales preocupaciones son la complejidad que tiene ya que esto ocuparía una gran cantidad en la memoria del equipo y por supuesto provocaría un tiempo de ejecución alto, pero en el caso de los algoritmos en paralelo, dado que se utilizan varios procesadores, el tiempo principal a optimizar es el que de comunicación entre procesadores. Esto depende de cómo se maneje el paralelismo en el lenguaje elegido para programar (Memoria compartida o Paso de mensajes).

Para modificar un algoritmo secuencial y convertirlo en paralelo, se debe evaluar parte por parte para ver la efectividad de paralelizarlo, ya que cierto procesamiento puede ser más eficiente manteniéndolo secuencialmente.

En este caso para el desarrollo del algoritmo de ataque se utilizó un diagrama de flujo donde se especificó variables y procesos, posteriormente se codificó en MATLAB. Para paralelizar se evaluó parte por parte, validando la eficiencia en ejecución secuencial y paralela, luego, encontrados los puntos de mejora se utilizó comandos de la librería PTC modificando y creando el algoritmo paralelo.

2. METODOLOGÍA

La presente aplicación tendrá las siguientes características:

- Una aplicación de uso didáctico en la que el usuario podrá realizar el cifrado, descifrado y ataque del S-DES, se tiene la disponibilidad de crear o subir un archivo de texto como entrada a cada una de las funciones mencionadas, de la misma manera al finalizar se podrá exportar el archivo o imprimirlo.
- En la opción de Ataque, se podrá seleccionar entre: serial o paralelo, dentro de la ejecución paralela se podrá especificar el número de workers que trabajarán en el proceso, mínimo 2 y máximo 21.

En la siguiente sección se detalla el desarrollo de los algoritmos utilizados para la implementación de la aplicación. Para cifrar y descifrar información se utilizará el algoritmo S-DES, para el ataque por fuerza bruta se construirá un algoritmo propio que cumpla con dicho objetivo, a continuación, se modificará de tal manera que pueda ser ejecutado en un ambiente paralelo y finalmente se explicará detalladamente como utilizar la computación paralela en MATLAB, para la construcción de un clúster en donde la aplicación podrá ser ejecutada.

2.1. ALGORITMO DE CIFRADO S-DES

El algoritmo de cifrado S-DES está compuesto por un conjunto de operaciones de permutación, sustitución, desplazamiento y por su puesto rondas Feistel como su esencia, a continuación, se describe paso a paso su funcionamiento.

2.1.1. RED FEISTEL

Algoritmo reversible, permite que su esquema de descifrado sea idéntico al cifrado únicamente con un cambio de orden de sub-claves. Consiste en un conjunto de operaciones que se repiten en cada ronda, el número de rondas depende del algoritmo en el cual será usada, para este caso (S-DES) se realizan 2 rondas.

Se trabaja con 2 datos de entrada: L_n y R_n (provienen de la división a la mitad de un vector de 8 bits), estos sufrirán varios cambios durante el proceso, en cada ciclo se utiliza una sub-clave, esta generada a partir de la clave inicial después de un proceso que se describirá en la siguiente sección dentro de las etapas del cifrado.

La **¡Error! No se encuentra el origen de la referencia.** describe las operaciones que se realizan dentro de la red de Feistel, plasmada gráficamente en la Figura 2.1.

$$R_I = L_{I-1} \oplus f(R_{I-1}, K_I) \quad (2.3)$$

Donde, K_i es el índice de la subclave, mientras que L_i y R_i son índices del bloque de mensaje.

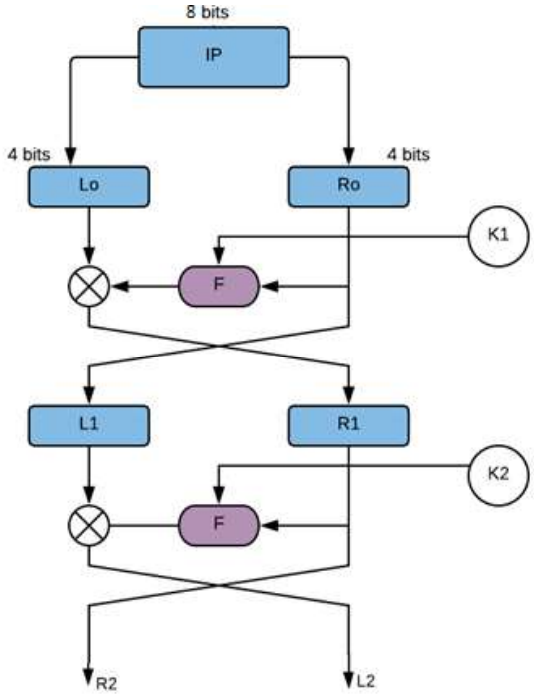


Figura 2.1. Red Feistel del S-DES.

2.1.2. ETAPAS DEL CIFRADO

El cifrado consta de 2 etapas, la primera consta de una secuencia de pasos para generar las sub-claves partiendo de la clave inicial y la segunda el cifrado en sí.

a. Generar sub-claves

El procesamiento de la clave K (10 bits) de cifrado sigue el esquema que muestra la Figura 2.2, como se puede ver, la entrada es una clave de 10 bits(vector), que pasa por un proceso de permutaciones, divisiones y desplazamientos hasta generar K1 y K2.

Para mejor comprensión de esta función se realizará un ejemplo a continuación, donde se detalla paso a paso el funcionamiento.

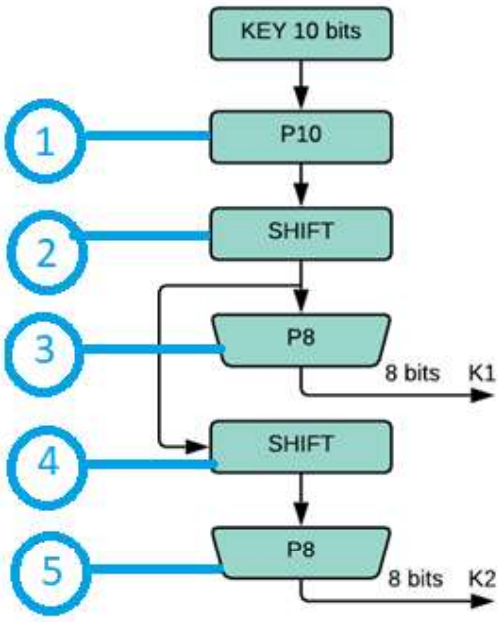


Figura 2.2. Generación de sub-claves K1 y K2.

Instrucciones
PASO 1 (figura 2.2) Se realiza un reordenamiento de acuerdo a la regla de generación del vector P10 a partir del valor de la clave la misma que será ingresada como argumento de entrada.(figura 2.3)

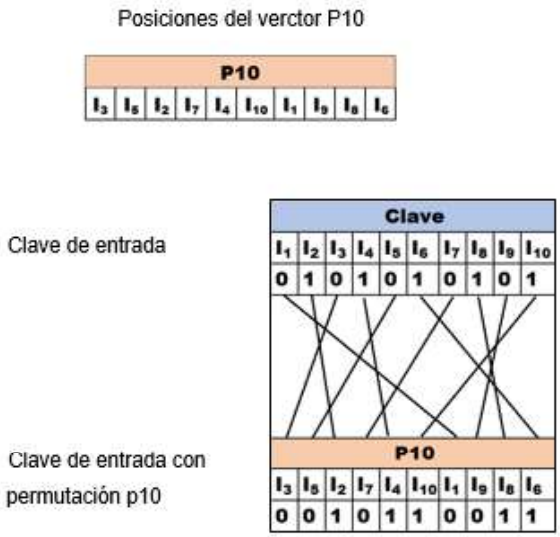


Figura 2.3. Permutación P10.

Código de MATLAB:

```
P10 = [llave(3) llave(5) llave(2) llave(7) llave(4) llave(10)
llave(1) llave(9) llave(8) llave(6)];
```

PASO 2 (figura 2.2)

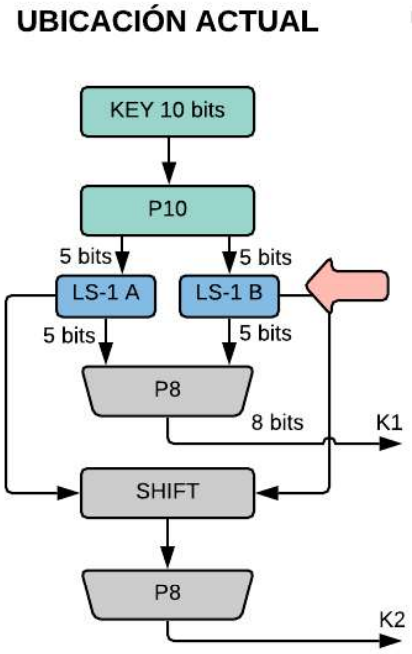


Figura 2.4. Desplazamiento a la izquierda.

Es necesario dividir a la mitad (figura 2.4)del vector P10 creado en el paso anterior, para esto se generan 2 nuevos vectores de longitud 5 bits, esta etapa consiste en recorrer el bit que está más hacia la izquierda y colocarlo más hacia la derecha, esto con cada vector de 5 bits (mitad1 y mitad2). Se forma un nuevo vector para cada caso (LS1_A, LS1_B) que iniciará desde el bit 2 de mitad1 o mitad2, en la posición final se colocará el bit b1 o b7.

Finalmente se procede a concatenar los resultados de LS1_A, LS1_B, obteniendo un vector de 10 bits que será útil en la siguiente etapa (figura 2.5).

Figura 2.5 Ejemplo de desplazamiento a la izquierda

Código de MATLAB:

```
mitad1 = P10(1:1:length(P10)/2);
mitad2 = P10((length(P10)/2)+1:1:length(P10));
LS1_A= mitad1(2:1:length(mitad1));
LS1_A = [LS1_A mitad1(1)];
LS1_B = mitad2(2:1:length(mitad2));
LS1_B = [LS1_B mitad2(1)]
```

PASO 3 (figura 2.2)

Aplicar la permutación P8 a continuación, se muestra cómo serán reordenados los bits, este resultado es el correspondiente a la primera sub-clave K1 (figura 2.6).

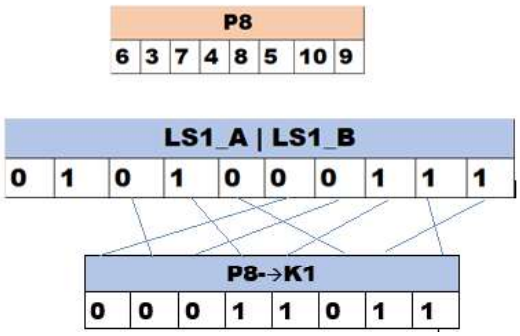


Figura 2.6 Obtenicón de la sub-clave k1.

Código de MATLAB:

```
union = [LS1_A LS1_B ];
P8 = [union(6) union(3) union(7) union(4) ...
union(8) union(5) union(10) union(9)];
K1=P8;
salida=K1;
```

PASO 4 (figura 2.2)

Se realiza el desplazamiento circular a la izquierda, pero en esta ocasión será de 2 bits.

Se forma un nuevo vector para cada caso (LS2_A, LS2_B) que iniciará desde el bit 3 hasta el final de mitad1 o mitad2, en las 2 últimas posiciones se colocarán los bits que fueron desplazados.

Finalmente se procede a concatenar los resultados de LS2_A, LS2_B, obteniendo un vector de 10 bits que será útil en la siguiente etapa. (figura 2.7)

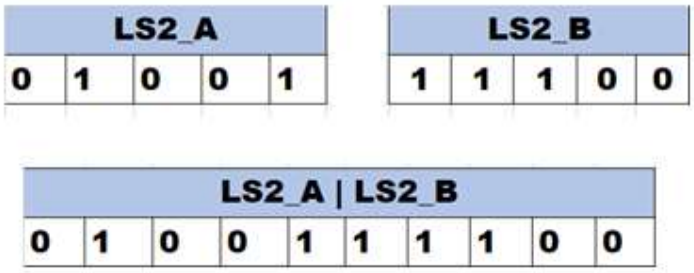


Figura 2.7 Desplazamiento a la izquierda 2 bits

Código de MATLAB:

```
LS2_A = LS1_A(3:1:length(LS1_A));
LS2_A = [LS2_A LS1_A(1) LS1_A(2)];
LS2_B = LS1_B(3:1:length(LS1_B));
LS2_B = [LS2_B LS1_B(1) LS1_B(2)];
```

PASO 5 (figura 2.2)

Aplicar la permutación P8 a continuación, se muestra cómo serán reordenados los bits, este resultado es el correspondiente a la segunda sub-clave K2 (figura 2.8).

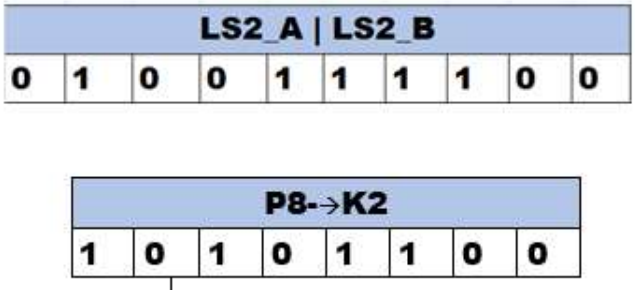


Figura 2.8 Obtenicón de la sub-clave k2.

Código de MATLAB:

```
union = [LS2_A LS2_B];
P8_fin = [union(6) union(3) union(7) union(4) ...
union(8) union(5) union(10) union(9)];
K2= P8_fin;
salida=K2;
```

b. Cifrado S-DES del texto claro.

El proceso de cifrado el texto plano que ingresa, es segmentado en bloques de 8 bits, que serán procesados uno a uno. El primer bloque pasará por la permutación inicial, ahora se aplica la primera ronda Feistel (F), a continuación, la segundo, con este resultado se aplica la permutación inversa y se obtendrá un bloque de texto de 8 bits cifrado.

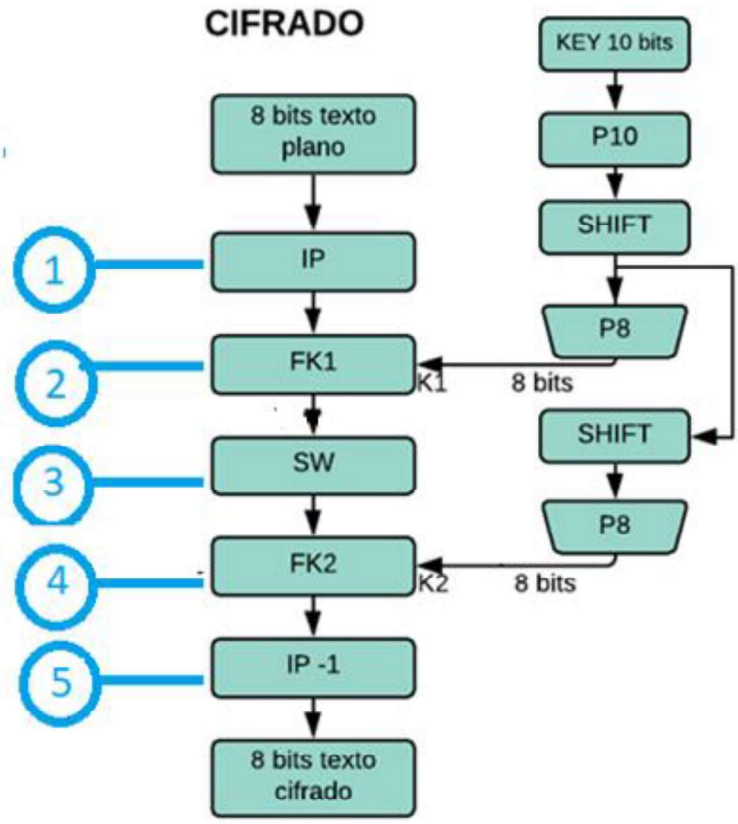


Figura 2.9. Diagrama de flujo del cifrado S-DES aplicado a cada uno de los caracteres del texto claro.

Para tener claridad de cómo funciona el proceso de cifrado se realizará un ejemplo, siguiendo el diagrama de flujo completo que se observa en la figura 2.9.

Instrucciones

Paso 1(figura 2.9)

Para el cifrado se necesita como argumento de entrada el texto claro, en este caso es el carácter “k”, se convierte a decimal y posteriormente a binario, se almacena en el array: textoVector, con este vector se aplica la permutación inicial IP, la cual consiste en el reordenamiento de las posiciones de los dígitos binarios. (figura 2.10)

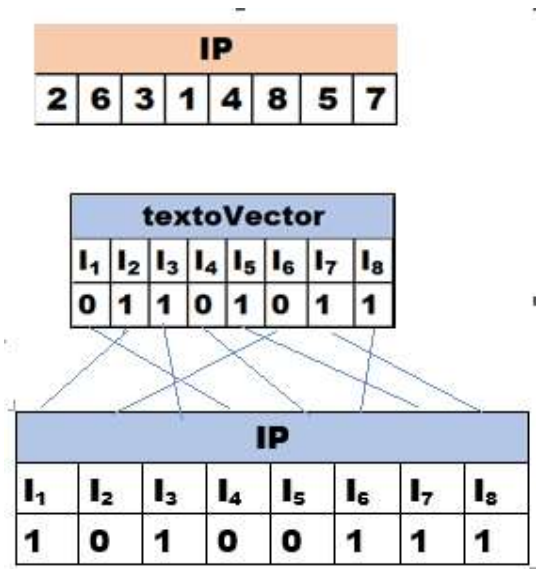


Figura 2.10. Permutación inicial a un carácter

Código de MATLAB:

```
valor = double (texto);
textoVextor=dec2bin(valor,8)

IP= [textoVector (2) textoVector(6) textoVector(3) textoVector(1)
textoVector (4) textoVector (8) textoVector (5) textoVector(7)];
```

Paso 2.1 (figura 2.9)

Se ejecuta la función fk con argumentos: vector IP y subclave K1.

Primera ronda Feistel, función F, toma como argumento de entrada el vector IP y la sub-clave K1, IP se segmenta en 2 vectores L y R, que corresponden a la parte izquierda y derecha de dicho vector. Se procesa R, este vector pasa por la permutación EP, cuyo objetivo es expandir de 4 bits a 8 con un reordenamiento y expansión (figura 2.11), en la figura 2.12 se tiene el resultado de EP respecto al ejemplo.

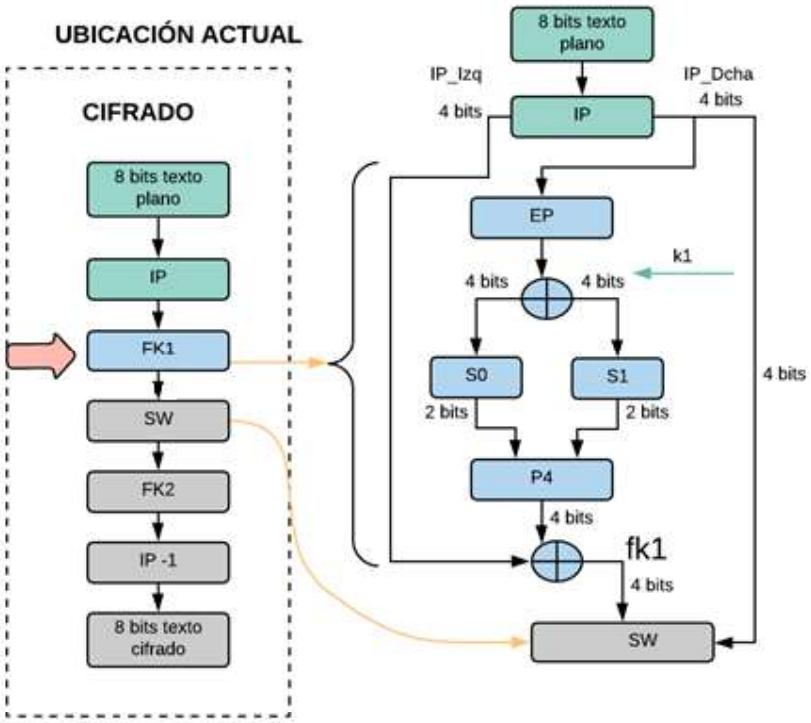


Figura 2.11. Funión fk1

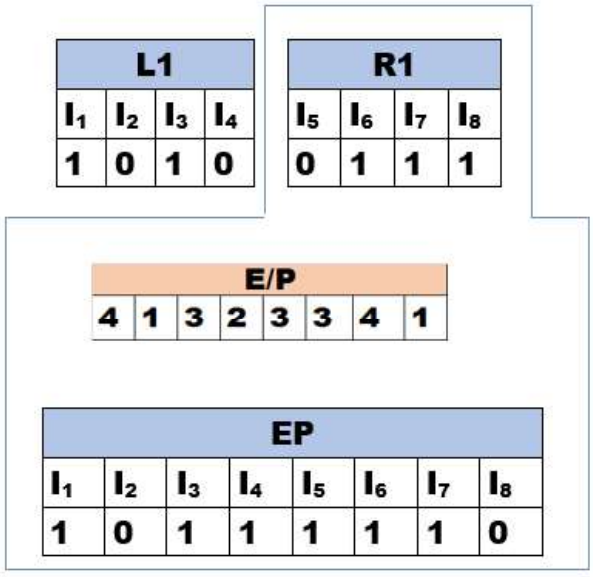


Figura 2.12. Permutación de expansión EP.

Código de MATLAB:

```
EP= [IP(8) IP(5) IP(6) IP(7) IP(6) IP(7) IP(8) IP(5)];
```

Paso 2.2 (figura 2.9)

La operación XOR, se realiza entre el vector EP y la sub-clave K1(figura 2.13), El resultado de la operación XOR, se descompone en dos partes iguales: Xor_1_A y Xor_1_B (figura 2.14)

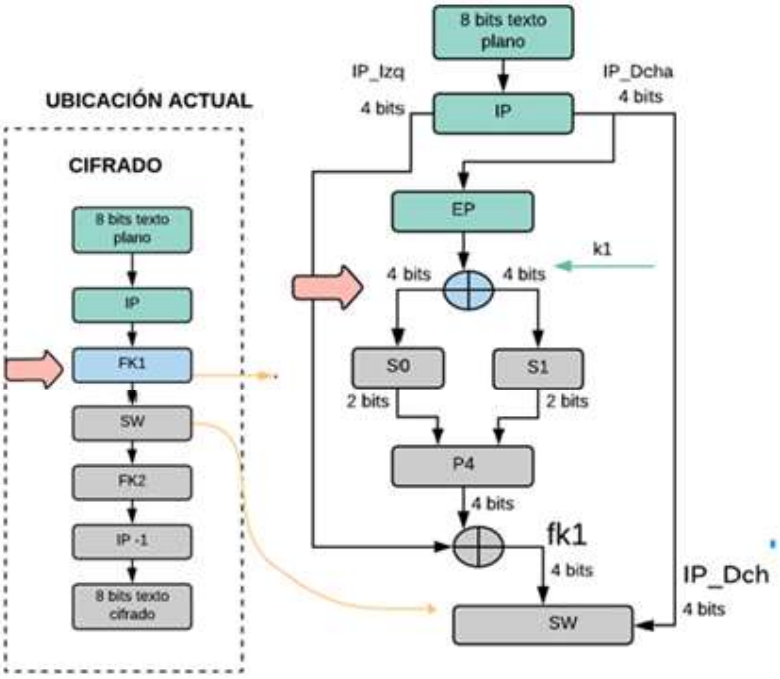


Figura 2.13. Operación XOR

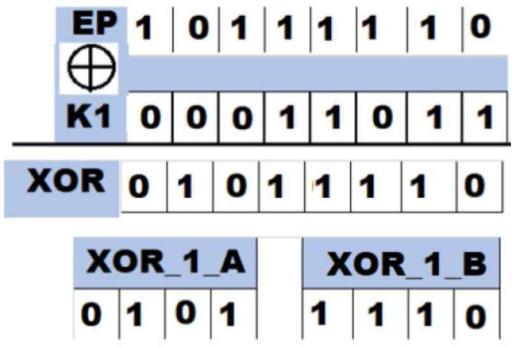


Figura 2.14. Ejemplo de operación XOR

Código de MATLAB:

```
Xor_1= xor (EP,K);
Xor_1_A= Xor_1 (1:1: length (Xor_1)/2);
Xor_1_B= Xor_1 (length (Xor_1)/2+1:1: length (Xor_1));
```

Paso 2.3 (figura 2.9)

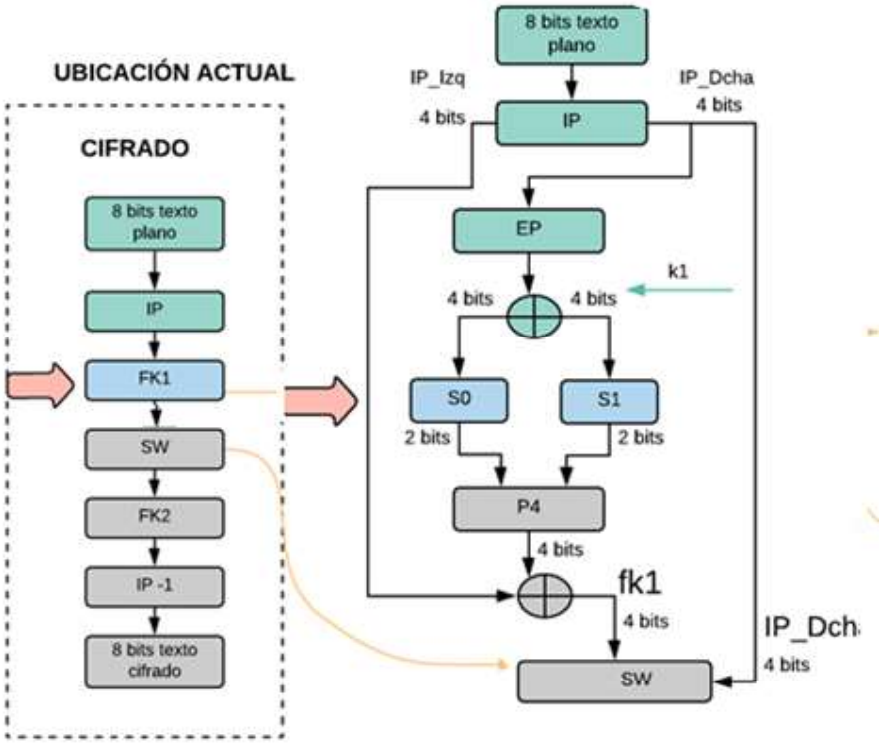


Figura 2.15. Ejemplo de operación XOR

Posterior a la división en segmentos del vector XOR_1(figura 2.15), se procede a la búsqueda de valores en las cajas S, estas se denominan cajas de sustitución, están compuestas por 4 filas y 4 columnas con valores binarios, van del 0 al 3, existe la caja S0 y S1.

Xor_1_A, servirá para realizar una consulta en S0 y Xor_1_B, servirá para realizar una consulta en S1. La búsqueda implica operaciones con los vectores mencionados.

El algoritmo indica que, para la búsqueda en So, se utilizará el bit 1 y 4 concatenándolos para buscar el valor obtenido en las filas para las columnas es el mismo proceso, pero con el bit 2 y 3(todo esto del vector Xor_1_A), se obtiene un valor binario de 2 bits.

El mismo proceso se realiza con S1, pero con el vector Xor_1_B, obteniendo otro valor binario de 2 bits.

El vector encontrado resultado de la caja S0 Y S1 se muestra en la figura 2.16.

S0	00	01	10	11
00	01	00	11	00
01	00	10	01	11
10	00	10	01	11
11	11	01	11	10

S1	00	01	10	11
00	00	01	10	11
01	10	00	01	11
10	11	00	01	00
11	10	01	00	11

Salida_S0_bin	
1	0

Salida_S1_bin	
0	1

Figura 2.16. Cajas s0 y s1

Código de MATLAB:

```
P00= Xor_1_A(1);
P01= Xor_1_A(2);
P02= Xor_1_A(3);
P03= Xor_1_A(4);

P10= Xor_1_B(1);
P11= Xor_1_B(2);
P12= Xor_1_B(3);
P13= Xor_1_B(4);

F_S0 = binvec2dec([P00 P03]);
C_S0 = binvec2dec([P01 P02]);

F_S1 = binvec2dec([P10 P13]);
C_S1 = binvec2dec([P11 P12]);

S0 = [1 0 3 2; 3 2 1 0; 0 2 1 3; 3 1 3 2];
F_S0 = F_S0 +1;
C_S0 = C_S0 +1;
Salida_S0 = S0 (F_S0, C_S0);
Salida_S0 = dec2binvec(Salida_S0,2);
Salida_S0_bin = [ Salida_S0 (2) Salida_S0 (1) ];
S1 = [0 1 2 3; 2 0 1 3; 3 0 1 0; 2 1 0 3];
F_S1 = F_S1 +1;
C_S1 = C_S1 +1;
Salida_S1 = S1 (F_S1, C_S1);
Salida_S1 = dec2binvec(Salida_S1,2);
Salida_S1_bin = [ Salida_S1 (2) Salida_S1 (1) ];
```

Paso 2.4 (figura 2.9)

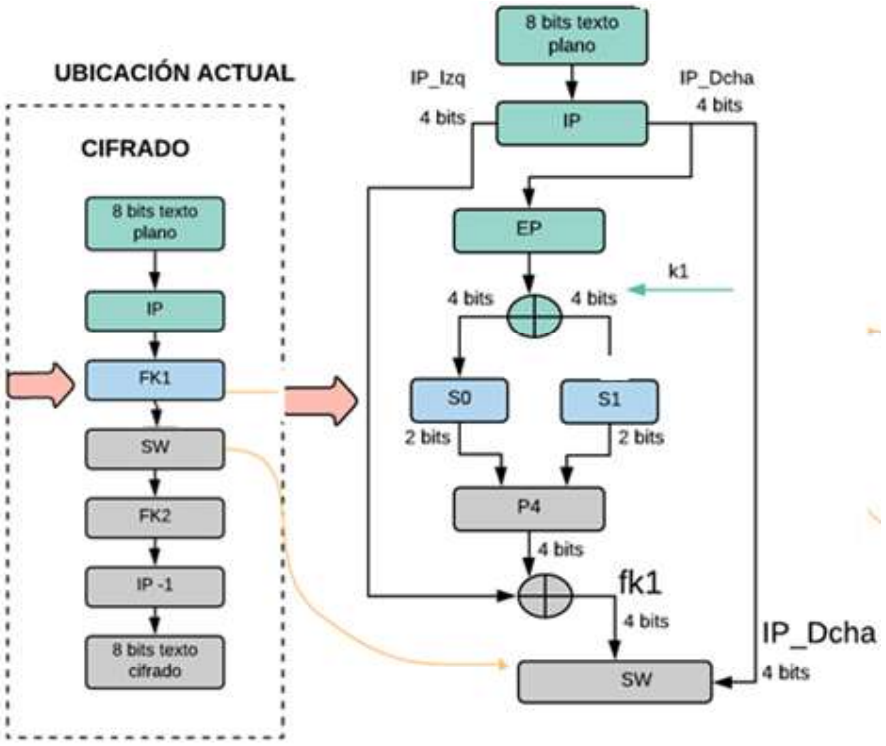


Figura 2.17. Permutación P4.

Se procede a concatenar las salidas y se aplica la permutación P4(figura 2.17), con la salida se aplica la función XOR con L1 que se obtuvo anteriormente (figura 2.18).

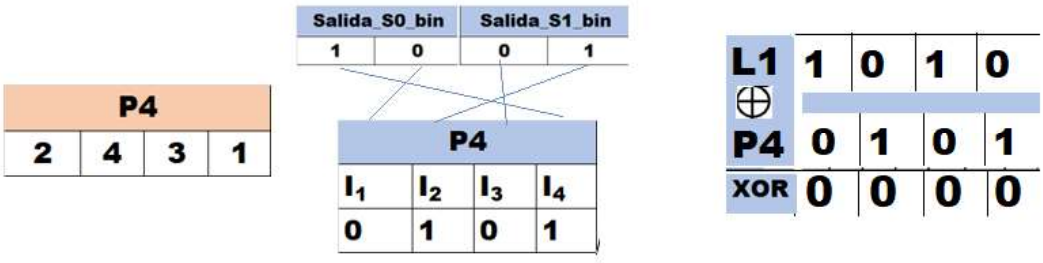


Figura 2.18. Ejemplo permutación P4 y XOR

Código de MATLAB:

```
union = [Salida_S0(2) Salida_S0(1) Salida_S1(2) Salida_S1(1)];
P_4=[union(2) union(4) union(3) union(1)];
result= xor(P_4,P_Iizq);
```

Paso 3 (figura 2.9)

Crear el vector de 8 bits “SW” concatenando los 4 bits de la derecha del vector IP(L1) y 4 bits generados en el bloque FK1 (figura 2.18).

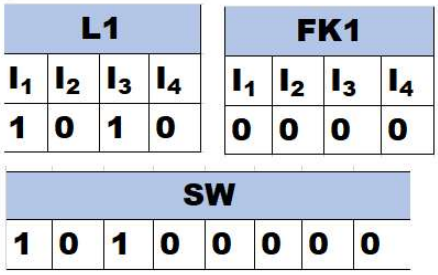


Figura 2.18. Vector Switch

Código de MATLAB:

```
SW=[R Fk1];
```

Paso 4 (figura 2.9)

Ejecución del bloque fk2. Se ejecuta la función fk con argumentos: vector “SW” y subclave K2. Se obtiene un vector de 4 bits resultado del proceso detallado en el paso 2 , luego se concatena con el vector L2 obteniendo el vector AUX (figura 2.18).

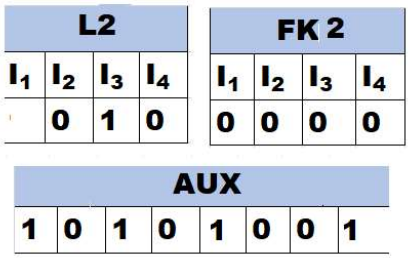


Figura 2.18. Función fk2

Código de MATLAB:

```
R2=[SW(5) SW(6) SW(7) SW(8)];
Fk2=fk(SW,K2);
aux=[Fk2 R2];
```

Paso 5 (figura 2.9)

El vector obtenido en el paso anterior pasa por la permutación inversa, una vez reordenados los bits, se obtiene el carácter cifrado.

Para obtener el valor en código ASCCI, se transforma el valor binario a decimal y luego a carácter, dando como resultado el valor cifrado= '×' (figura 2.19).

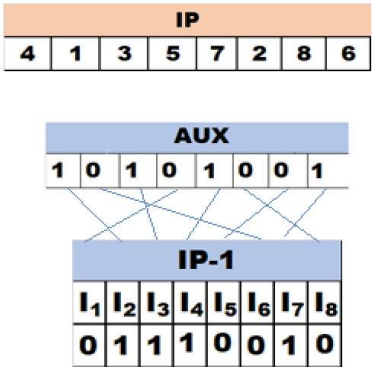


Figura 2.19. Permutación inversa.

Código de MATLAB:

```
PIinversa= [aux(4) aux(1) aux(3) aux(5) aux(7) aux(2) aux(8)
aux(6)];
pp1=bi2de(PIinversa);
cifrado=char(pp1);
```

Este proceso mostró el cifrado de unicamente un carácter; sin embargo, se va utilizar un archivo de texto que contenga un conjunto de caracteres, por lo que se agrega el siguiente segmento de código para obtener el argumento de entrada seleccionado:

```
[fileName, Path]=uigetfile({'*.txt'},'Abrir Documento de
Texto');

texto_1=fopen(fileName, 'r', 'n');

texto=fread(texto_1);

fclose(texto_1);
```

Segmento de código 2.1. Abrir y leer un archivo de texto.

La función se modifica dentro de un ciclo for para poder cifrar un conjunto de carácteres (Segmento de código 2.2).

```
function [cifrado]=cifrar(texto,K1,K2)
valor = double (texto);

for hh=1:length(valor)
% El valor ASCII decimal se pasa a un vector tipo binario de 1 x
      8
textoVector = dec2binvec (valor(hh),8);
% Permutación inicial
IP=[textoVector(2) textoVector(6) textoVector(3) textoVector(1)
      ...
textoVector(4) textoVector(8) textoVector(5) textoVector(7)];
R=[IP(5) IP(6) IP(7) IP(8)];
% FK1
Fk1=fk(IP,K1);
SW=[R Fk1];
%FK2
R2=[SW(5) SW(6) SW(7) SW(8)];
Fk2=fk(SW,K2);
aux=[Fk2 R2];
PIinversa= [aux(4) aux(1) aux(3) aux(5) aux(7) aux(2) aux(8)
      aux(6)];
pp1=bi2de(PIinversa);
cifrado(hh)=char(pp1);
end
end
```

Segmento de código 2.2. Función cifrado S-DES

2.2. ALGORITMO DE DESCIFRADO S-DES

El descifrado de S-DES es metódicamente el mismo que el de cifrado, por su puesto la entrada será un texto cifrado y la clave será la misma con la que dicho texto fue cifrado. El proceso se muestra en la figura 2.20, donde se observa que en la primera ronda Feistel ingresa como entrada la sub-clave K1 y en la segunda la sub clave K2, siendo esto lo único que lo diferencia del cifrado.

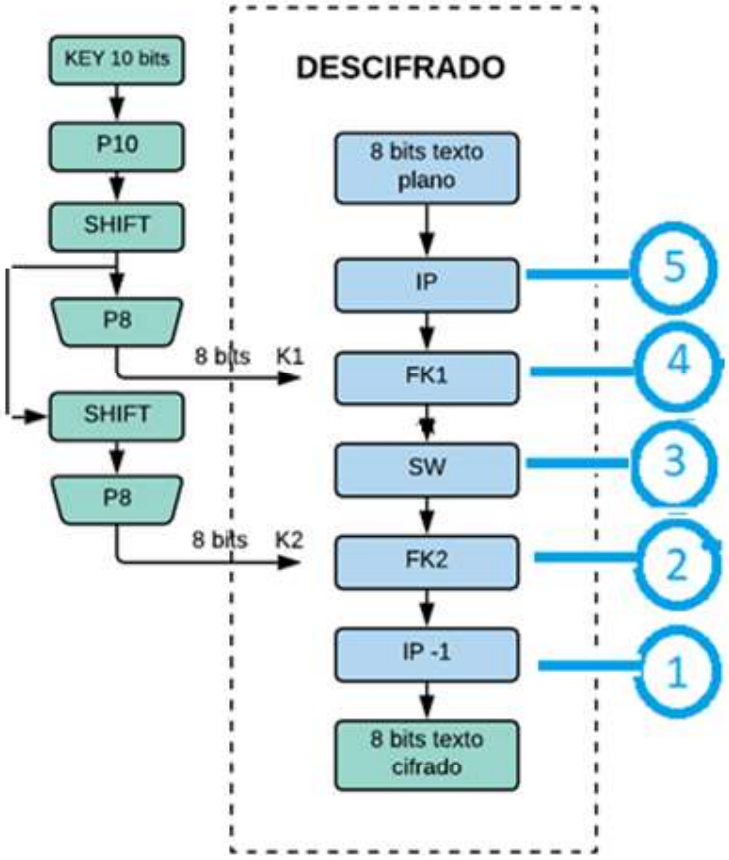


Figura 2.20. Diagrama de flujo del descifrado S-DES

Instrucciones
Paso 1 (figura 2.20) El proceso de descifrado es el mismo que el cifrado, únicamente invirtiendo el cifrado, lo primero que se realiza es la transformación a binario del texto cifrado. Se realiza la permutación inicial de acuerdo a la regla de generación del vector PI

Código de MATLAB:
`textoVector_d = dec2binvec (double(cifrado),8)`
`PIinversa=[aux(4) aux(1) aux(3) aux(5) aux(7) aux(2) aux(8) aux(6)];`

Paso 2 (figura 2.20) Ejecución del bloque fk2. Se ejecuta la función fk con argumentos: vector IP y subclave K2, de lo cual se obtiene como resultado 4 bits de las cajas fijas S0 y S1 del algoritmo.

Código de MATLAB:

```
Fk2=fk(IP,K2);
```

Paso 3 (figura 2.20)

Creación del vector de 8 bits “SW” con 4 bits de la derecha del vector IP y 4 bits generados en el bloque FK2.

Código de MATLAB:

```
IP_Dcha=[IP(5) IP(6) IP(7) IP(8)];
SW=[IP_Dcha Fk2];
```

Paso 4 (figura 2.20)

Ejecución del bloque fk1. Se ejecuta la función fk con argumentos: vector “SW” y subclave K1, del cual se obtiene 4 bits de las cajas fijas S0 y S1 , luego se crea el vector de 8 bits “aux” con 4 bits de la derecha del vector “SW” y 4 bits generados en el bloque FK1.

Código de MATLAB:

```
Fk1=fk(SW,K1);
IP_Dcha_2=[SW(5) SW(6) SW(7) SW(8)];
aux=[Fk1 IP_Dcha_2];
```

Paso 5(figura 2.20)

Ejecución de la permutación inversa inicial, de acuerdo a la regla de generación de vector PI-inversa a partir del vector “aux”.

Transformación de binario a decimal (ascii) y Transformación de decimal (ascii) a carácter.

Código de MATLAB:

```
IP=[textoVector_d(2)      textoVector_d(6)      textoVector_d(3)
textoVector_d(1)        textoVector_d(4)        textoVector_d(8)
textoVector_d(5) textoVector_d(7)];
pp_2=bi2de(PIinversa);
solu=char(pp_2);
```

La función se modifica dentro de un ciclo for para poder cifrar un conjunto de caracteres (Segmento de código 2.3).

```
function [solu]=descifrarSerial(cifrado,K1,K2)
for hh=1:length(cifrado)
            % Permutación inicial
            textoVector_d = dec2binvec
(double(cifrado(hh)),8);
            IP=[textoVector_d(2) textoVector_d(6)
textoVector_d(3) textoVector_d(1) ...
            textoVector_d(4) textoVector_d(8)
textoVector_d(5) textoVector_d(7)];
            IP_Dcha=[IP(5) IP(6) IP(7) IP(8)];
            Fk2=fk(IP,K2);
            SW=[IP_Dcha Fk2];
            Fk1=fk(SW,K1);
            IP_Dcha_2=[SW(5) SW(6) SW(7) SW(8)];
            aux=[Fk1 IP_Dcha_2];
            PIinversa=[aux(4) aux(1) aux(3) aux(5) aux(7)
aux(2) aux(8) aux(6)];
            aux=bi2de(PIinversa);
            solu(hh)=char(aux);
end
end
```

Segmento de código 2.3.. Función descifrado S-DES

2.3. PROBLEMA AL RECUPERAR EL CONTENIDO DEL ARCHIVO CIFRADO

Se tiene como base comparativa el texto en claro (figura 2.21) el resultado del cifrado es en su mayoría un conjunto de caracteres especiales(figura 2.22), cuando este valor se guardó en el archivo, no sigue una codificación específica por lo cual al momento de leer el archivo en el script de descifrado el valor no corresponde al original, ciertos caracteres cambian su valor, lo cual provoca un descifrado incorrecto, 1 o 2 caracteres fallan, de forma que cuando se le el archivo de texto en claro cambiará por completo el sentido del mismo (figura 2.23).

En criptografía, se denomina ataque de fuerza bruta a la forma de recuperar una clave probando todas las combinaciones posibles hasta encontrar aquella que permite el acceso, el texto puede tener números(1234) y símbolos(·#□).

Figura 2.21. Texto plano a ser cifrado con el S-DES

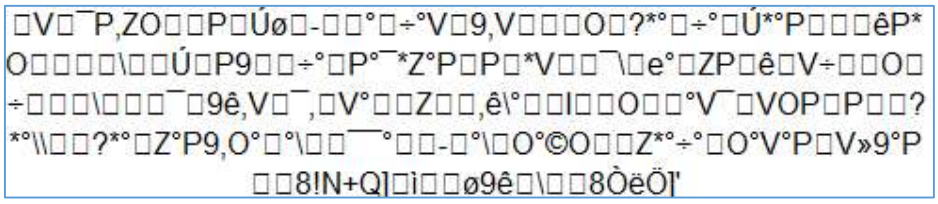


Figura 2.22. Texto ya cifrado con el S-DES.

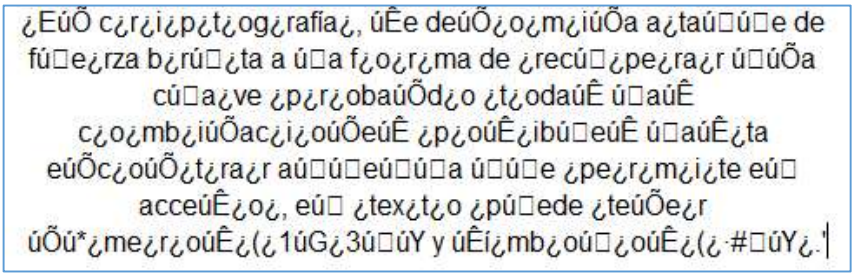


Figura 2.23. Texto obtenido luego de descifrarlo con el S-DES

El problema radica en la función fopen:

*fileID = fopen(filename,permission,machinefmt,**encodingIn**)*

- Permission: es el tipo de acceso al archivo, en este caso se requiere permisos de lectura, por lo que se utiliza 'r'.
- Machinefmt : indica el orden de lectura de los bytes que forman el archivo de texto. Se configura para que utilice el valor por defecto con 'n'.
- **EncodingIn:** esquema de codificación, si no se especifica, abre el archivo para su procesamiento usando la codificación predeterminada en el sistema, en este caso ASCCI de 8 bits.

La solución se basa en la codificación UTF-8:

UTF-8 es una codificación de caracteres de ancho variable, capaz de codificar todos los 1,112,064 válidos en Unicode utilizando de uno a cuatro bytes. La codificación está definida por el estándar Unicode, y fue originalmente diseñada por Ken Thompson y Rob Pike. El nombre se deriva del formato de transformación Unicode (o juego de caracteres codificado universal) - 8 bits .

Fue diseñado para compatibilidad con ASCII. La principal diferencia es que un carácter ASCII puede caber en un byte (8 bits), pero la mayoría de los caracteres Unicode no pueden.

El hecho de ser Unicode, significa que se acopla con varios idiomas y por su puesto es compatible con ASCII, de modo que es más común utilizar caracteres no latinos y esto no lo permite ASCII.

Características:

- 1 byte: ASCII estándar
- bytes: árabe, hebreo, la mayoría de los guiones europeos (más notablemente excluyendo a los georgianos)
- bytes: BMP
- bytes: todos los caracteres Unicode

Se realiza 2 cambios, en el script de cifrado y en el de descifrado.

Especificar en la parte encoding de la función Fopen al momento de guardar el resultado del cifrado UTF-8 un tipo de codificación que es compatible con ASCII, esta le indicará de qué manera leer cada carácter.

```
fid=fopen('textoCifrado.txt','r/w','n','UTF-8');
```

Cambiar la función char() que maneja una codificación ASCII, por la función native2unicode(bytes, 'UTF-8'), cuyos argumentos de entrada son:

- Texto recuperado después de leer el archivo que estará en decima y la codificación.
- Tipo de codificación UTF-8 con se guardó dicho archivo.

El resultado es un vector de nx1, donde n será la longitud del texto cifrado, posterior a esto se le aplica la traspuesta a este vector para procesar caracteres 1 por 1.

Se obtiene un mensaje de éxito (figura 2.24), por lo cual se comprueba el funcionamiento correcto del descifrado utilizando el S-DES.

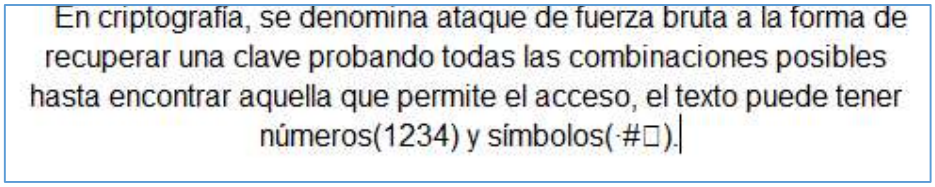


Figura 2.24. Texto obtenido al descifrarlo con el S-DES, luego de realizar los cambios respecto de la opción UTF-8

2.4. CONSTRUCCIÓN DEL ALGORITMO DE ATAQUE POR FUERZA BRUTA AL S-DES

Para construir el algoritmo de ataque se analizó el comportamiento del texto cifrado y descifrado.

En el caso de cifrar un texto, cualquier sea su tamaño, dado un cifrado simétrico, lo más importante es la clave con la que se cifró, por lo cual, la idea central es barrer todo el espacio de claves que en este caso es 2^{10}= 1024 posibilidades, de manera que una de estas encuentre el texto original o texto claro.

Se han considerado 2 posibilidades para desarrollar dicho algoritmo:

Posibilidad 1: generando al inicio la matriz de claves.

Posibilidad 2: sin generar matriz de claves.

2.4.1. POSIBILIDAD 1:ETAPAS DEL ALGORITMO DE ATAQUE GENERANDO MATRIZ DE CLAVES

El desarrollo del algoritmo consta de 3 etapas: a) generación de claves posibles, b) descifrado con cada clave y c) Filtrado de resultados.

a) Generación de la matriz de claves posibles.

Dentro de esta etapa se genera un matriz que contiene todas las posibles claves necesarias para descifrar un texto.

La clave contiene 10 bits **(**Figura 2.25), por lo tanto, el número de posibilidades es 2^{10}-1=1023, donde las posibilidades van a variar desde todos los bits con 0, hasta todos los bits con 1.

Llave									
B1	B2	B3	B4	B5	B6	B7	B8	B9	B10

Figura 2.25. Clave para cifrado y descifrado S-DES.

El espacio de claves a recorrer para encontrar la correcta es:

$$Ki = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Figura 2.26. Espacio de claves para evaluar en el ataque por fuerza bruta.

Entonces las claves posibles van desde 0 hasta 1023, por lo que se establece una matriz 1023X10 (Figura 2.26).

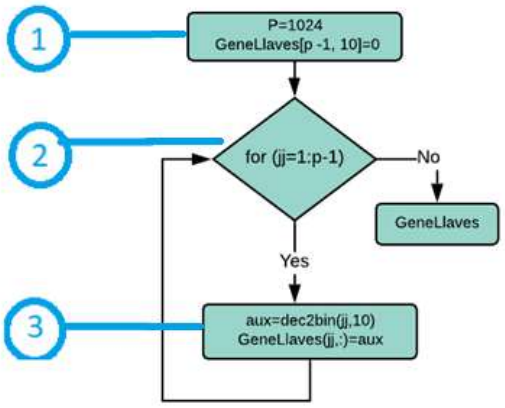


Figura 2.27. Diagrama de flujo para generar la matriz de claves posibles.

Para una mejor comprensión, a continuación, se explicará a detalle el proceso para generar una matriz de claves posibles como lo indica el diagrama de flujo de la figura 2.27.

Instrucciones

Paso 1 (figura 2.27)

Se establece una variable que almacena el valor de posibilidades, para generar una matriz de ceros con la longitud que se especifica mediante la variable p=1024 (figura 2.28).

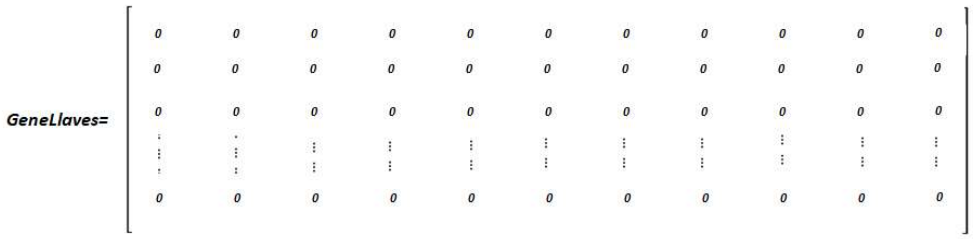


Figura 2.28. Matriz vacia de dimensión 1023x10

Código de MATLAB:

```
p=2^ (10);
GeneLlaves=zeros(p-1,10);
```

Paso 2 (figura 2.27)

Para llenar la mencionada matriz, de nombre GeneLlaves, se utiliza un ciclo for, desde 0 hasta 1023, donde se recorrerá fila por fila, llenado cada una por el valor binario de la variable jj, la cual va contando uno por uno, mediante la función dec2binvec, cuya longitud del vector que retorna deberá ser de 10 bits, por el tamaño fijo que denota el S-DES para la clave, la función fliplr causa un efecto espejo en los bits de la fila, para que las claves sigan el orden, sumando 1 en decimal.

Código de MATLAB:

```
for jj=1:p-1
    GeneLlaves(jj,:)=fliplr(dec2binvec(jj,10));
end
```

Paso 4 (figura 2.27)

Se obtiene una matriz de dimensiones 1023x10 llena con todo el espacio de claves posibles (figura 2.29).

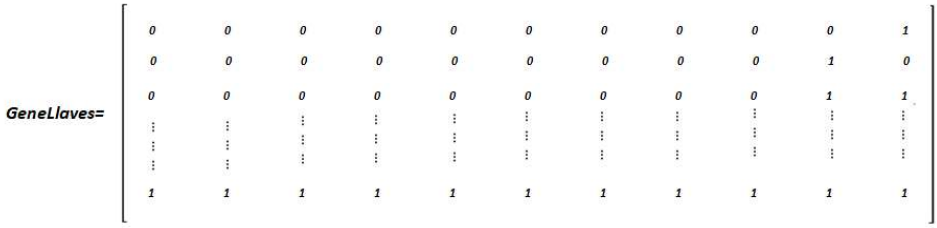


Figura 2.29. Matriz de posibles claves

b) Descifrado con cada clave

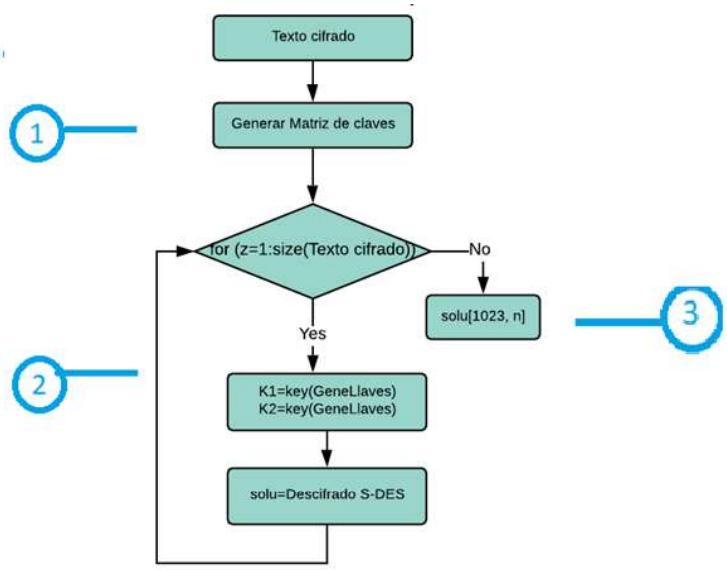


Figura 2.30. Diagrama de flujo que descifra un texto por clave.

Para descifrar los posibles textos, se sigue el proceso que indica el diagrama de flujo de la figura 2.30. Los textos generados serán 1023 de igual manera que las claves, así que se especifica una matriz para que los guarde, el número de columnas dependerá de la longitud

del texto cifrado. Los sub procesos: Generar Matriz de claves, K1, K2 y descifrar, se encuentran detallados en las Figuras 2.39, 2.2 y 2.36 respectivamente.

Mientras más grande sea el texto, más tiempo tomará el intento para descifrar, y evaluar si la clave fue correcta.

Para una mejor explicación, se utiliza un ejemplo estableciendo como argumento de entrada un texto pequeño.

Instrucciones

Paso 1 (figura 2.30)

Texto en claro: "En la criptografía, el cifrado" al que se desea llegar a partir del texto cifrado: "ž ²Uœ.U$ßëñÄ÷ß.Tv.HU;œU$ëTß.^÷"

Con clave desconocida, será segmentado en caracteres.

A continuación se genera la matriz de posibles claves (figura 2.31).

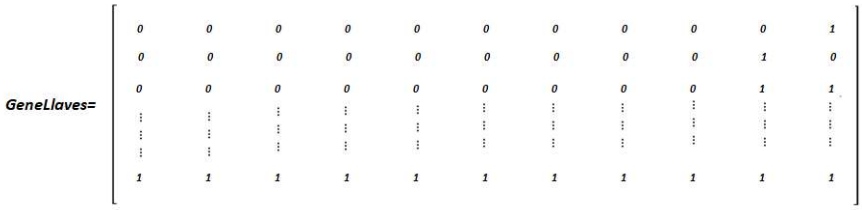


Figura 2.31. Matriz de posibles claves

Código de MATLAB:

```
p=2^(10);
GeneLlaves=zeros(p-1,10);
for jj=1:p-1
    GeneLlaves(jj,:)=fliplr(dec2binvec(jj,10));
end
```

Paso 2 (figura 2.30)

Se utiliza un bucle que permite tomar cada carácter del texto y cada fila de la matriz GeneLlaves, este valor de 10 bits primero pasará como argumento de entrada para la función keys, así se obtendrá las sub-claves K1 y K2.

Luego se procede a descifrar el texto, este proceso se repite 1023 veces, se crea una matriz para que guarde los textos descifrados, el número de columnas dependerá de la longitud del texto cifrado.

Código de MATLAB:

```
for z=1:size(GeneLlaves,1)
    K1=keys(GeneLlaves(z,:),1);
    K2=keys(GeneLlaves(z,:),2);
   Solu(z,:)=descifrarSerial(ParteCifrado,K1,K2);
end
```

Paso 3 (figura 2.30)

Con la finalización del ciclo anterior, se obtiene una matriz cuya longitud es de 1023x26, debido a la longitud del texto cifrado. Dicho arreglo contiene en una de sus filas el resultado requerido: texto en claro, pero ¿cómo saber cuál es?, se discutirá en la siguiente sección (figura 2.32).

$$solu = \begin{bmatrix} cc1 & cc2 & cc3 & cc4 & cc5 & cc6 & \dots & \dots & \dots & cc226 \\ cf2 & cf3 & cf4 & cf5 & cf6 & cf7 & \vdots & \vdots & \vdots & \vdots \\ cf3 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ cf4 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ cf5 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ cf6 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ cf1023 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

Figura 2.32. Matriz de textos descifrados

c) Filtrado de resultados

Al aplicar cada clave obtenida al proceso de descifrado, se obtendrá 1023 posibilidades de textos.

Para elegir el texto correcto, se analiza cómo está constituida la matriz que almacena los textos descifrados con todas las posibilidades de claves generadas. Para una comprensión más sencilla, se analiza un segmento de dicha matriz.

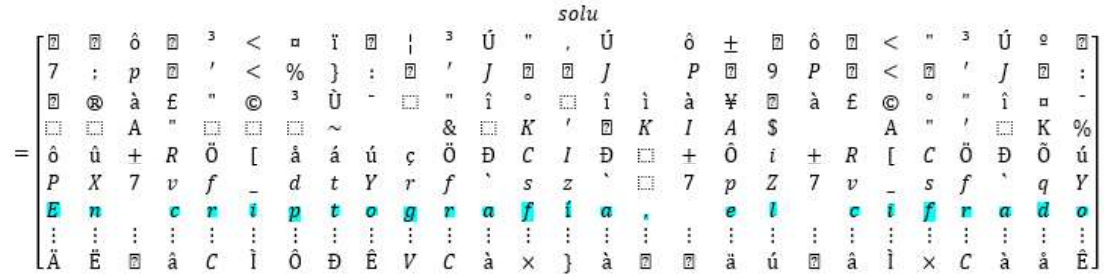


Figura 2.33. Matriz de textos descifrados obtenida.

Como se puede ver en un análisis fila por fila, en las 3 primeras, los caracteres que forman el vector tienen en su mayoría caracteres especiales, en la cuarta y quinta fila pasa lo mismo; sin embargo, aquí ya existen ciertos caracteres comunes consonantes y vocales; en la sexta además de lo mencionado existen ciertos números, lo común entre estas es que el texto que forman no tiene un sentido, no forman palabras ni frases. En la fila 7 que se encuentra el texto con la clave correcta, se puede observar que el carácter espacio se encuentra con una frecuencia mayor que todas las letras y no existen caracteres especiales, lo cual lo diferencia de las primeras filas de la matriz y de las siguientes (figura 2.33).

Estrategias para obtener el texto en claro

Se propone evaluar 2 opciones respecto a los caracteres que conforman un texto claro, con el objetivo de tener éxito en el ataque. Para analizar los resultados, se evalúa un texto de aproximadamente 1 páginas.

- Opción 1: contar caracteres alfanuméricos y espacios.
- Opción 2: contar caracteres especiales (extendidos y de control),

ASCII control characters		
00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters					
32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Extended ASCII characters							
128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

Figura 2.34. Código ASCII

- **Opción 1**: Contar caracteres alfanuméricos y espacios.

Mediante el análisis de obras literarias bastante extensas como La Regenta [32] y Don Quijote de la Mancha [33], se han obtenido frecuencias asignadas a la repetición de caracteres:

Para la primera obra se ha conseguido los siguientes resultados:

-Caracteres más frecuentes: espacio, seguido por la “a”.

Para la segunda obra se obtienen resultados un poco diferentes:

-Carácter más frecuente: ‘e’.

Por otro lado, según la RAE se considera los siguientes hechos:

-El carácter más frecuente es la “a” con un porcentaje superior al 13%

-Las vocales conforman hasta el 45% de un texto

En ambas obras se cumple que la suma de frecuencias de las vocales supera el 45% del total, coincidiendo con el indicador de la RAE, además que la suma de las frecuencias de los caracteres especiales, no supera el 10% en ninguno de los casos.

Estos resultados se ha utilizado para la toma de decisiones al momento de condicionar el filtrado de la siguiente manera:

El texto se evalúa según los valores decimales ASCII de las letras mayúsculas (65 al 90), minúsculas (97 al 122), números (48 al 57) y finalmente el carácter espacio cuyo valor es 32. Para mejor comprensión, la figura 2.16 detalla la clasificación de la codificación ASCII. El resultado es un vector cuyo número de columnas aumenta conforme un carácter cumpla la condición especificada, por lo cual es necesario obtener la longitud de dicho vector para obtener la cantidad de caracteres que estén dentro del rango especificado.

```
count(z)=length(find((solu>=65 & solu<=90) | (solu>=97 & solu<=122)|
(solu>=48 & solu<=57) | solu==32));
```

Ahora es preciso calcular el porcentaje con el dato obtenido en el paso anterior y comparar con un valor base, el cual se ha considerado cerca del 90% tomando como base el análisis de las obras literarias mencionadas. Este valor además es válido ya que, si se examina un texto con sentido en el idioma español, este tiene un bajo porcentaje de caracteres especiales y por ello se deja un margen del 10% en caso que el texto exista correos electrónicos, si un texto la cumple, se trata del texto en claro verdero.

```
if ((count(z)/length(a)) *100>90)
```

- **Opción 2:** Contar caracteres especiales

Con base a este criterio se descartará un texto si su contenido supera en un 10% caracteres especiales, criterio en base a las premisas presentadas en un inicio, revisar figura 2.34 donde están los caracteres con sus valor decimal, para esta condición se especifica desde el valor 1 al 31 que corresponden a los caracteres de control, 33 al 43 parte de caracteres imprimibles, no se toma en cuenta del 44 al 47 donde se encuentran “.,/-” y finalmente del 123 al 255.

```
count=length(find((solu>=123&solu<=255)|(solu>=1&solu<=31)|(solu>=
33&solu<=43)));
```

```
if ((count(z)/length(a))*100>5)
```

Para comprobar el funcionamiento del algoritmo se eligen 2 textos: texto corto de aproximadamente 10 líneas y texto largo: aproximadamente 3 páginas en la peor condición, es decir con la última clave de la matriz “1111111111”.

La eficiencia para detectar la clave correcta no solo depende de los criterios para la obtención del texto sino también de la dificultad de la clave, para este caso se cifró con una clave cuyo valor decimal es el 8, de tal manera que los tiempos serán cortos.

Al probar el funcionamiento de esta estrategia con diferentes claves, se encuentra que con ciertas claves no solo un texto cumple dicha condición, ya sea la de caracteres normales o especiales, se procede analizar el resultado erróneo que cumple la condición, se considera el texto corto: “la criptografía es” y la clave “1000000001”.
Al momento de ejecutar el ataque, a la salida se obtiene que la clave correcta es: “1111110101” y el texto supuestamente correcto: “lA cfpth0gr” fga eh gttjciyhvncDiA”
Se observa que los caracteres que conforman el texto descifrado son comunes, los espacios están separando palabras, algo que se puede resaltar es que hay demasiadas consonantes lo cual hace que no tenga sentido, se toma la premisa que indica que el porcentaje de vocales corresponde hasta un 45%(modificada tomando en cuenta varias pruebas), realizando una combinación con la existente.
La variable count contiene el número de caracteres especiales en el texto:

```
count=(length(find((solu>=123 & solu<=255)|(solu>=1&
solu<=31)|(solu>=33& solu<=43))))/length(a);
```

La variable count1 contiene el número de vocales en el texto:

```
count1=(length(find(solu==97|solu==101|solu==105|solu==111|solu==1
17)))/length(a);
```

La condición queda establecida con un porcentaje menor al 10% en caracteres especiales y mayor al 25% en vocales, como se detalla en el diagram de flujo en la figura 2.35.

El proceso de filtrado tomando en cuenta la matriz resultado que contiene todos los textos descifrados se ha implementado en MATLAB y se detalla dentro del bucle principal del algoritmo y se detalla en el Segmento de código 2.4.

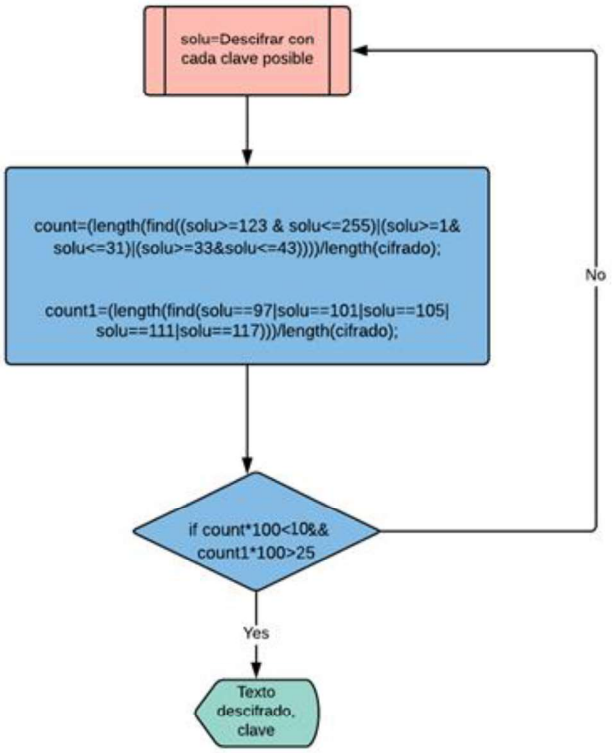


Figura 2.35. Diagrama de flujo del proceso de filtrado de texto correcto.

```
solu=descifrar(ParteCifrado,K1,K2);

count=(length(find((solu>=123   &   solu<=255)|(solu>=1&
solu<=31)|(solu>=33& solu<=43))))/length(ParteCifrado);
count1=(length(find(solu==97|solu==101|solu==105|solu==
111|solu==117)))/length(ParteCifrado);

if count*100<3 && count1*100>25

if length(cifrado)<80
disp(solu(z,:));
disp(GeneLlaves(z,:));
else
solu=descifrarSerial(cifrado,K1,K2);
           disp(solu(z,:));
disp(GeneLlaves(z,:));
end

break;
end
```

Segmento de código 2.4. Filtrado del texto correcto.

A continuación se prosigue con el ejemplo anterior a partir de la matriz de posibles textos claros, utilizando el filtrado.

Instrucciones

En base a la figura 2.35:
Dada la matriz “solu” que contiene todos los posibles resultados, se verifica fila por fila si cumple con la condición del filtrado, si es así se encuentra el texto correcto, de otra manera se continua recorreindo las filas disponibles hasta cumplir dicha condición.

En este caso en la fila 7 ya se encontró lo requerido, se procede almacenar el texto y la clave (figura 2.36).

Figura 2.36. Texto original encontrado.

Código de MATLAB:

```
for z=1:size(GeneLlaves,1)
    K1=keys(GeneLlaves(z,:),1);
    K2=keys(GeneLlaves(z,:),2);
   solu=descifrarSerial(ParteCifrado,K1,K2);

count=(length(find((solu>=123&solu<=255)|(solu>=1&solu<=31)|(sol
u>=33& solu<=43))))/length(ParteCifrado);

count1=(length(find(solu==97|solu==101|solu==105|solu==111|solu=
=117)))/length(ParteCifrado);

    if count*100<10 && count1*100>25
```

Todo lo especificado como parte del algoritmo de ataque se encuentra en el Segmento de código 2.5.

```
function [llave,texto]=ataqueSerial(cifrado,ParteCifrado)
p=2^(10);
GeneLlaves=zeros(p-1,10);


for jj=1:p-1
    GeneLlaves(jj,:)=fliplr(dec2binvec(jj,10));
end
for z=1:size(GeneLlaves,1)
    K1=keys(GeneLlaves(z,:),1);
    K2=keys(GeneLlaves(z,:),2);

   solu=descifrarSerial(ParteCifrado,K1,K2);
  count=(length(find((solu>=123 & solu<=255)|(solu>=1&
solu<=31)|(solu>=33& solu<=43))))/length(ParteCifrado);

count1=(length(find(solu==97|solu==101|solu==105|solu==111|solu==
117)))/length(ParteCifrado);

    if count*100<10 && count1*100>25

        if length(cifrado)<80
   disp(solu(z,:));
   disp(GeneLlaves(z,:));
        else
           solu=descifrarSerial(cifrado,K1,K2);

        end

       break;
    end
end

llave=z;
texto=solu;
end
```

Segmento de código 2.5. Algoritmo de ataque al S-DES

2.4.2. POSIBILIDAD 2: SIN GENERACIÓN DE MATRIZ DE CLAVES

Un posible problema del método anterior es que la generación de la matriz de claves puede ocasionar una carga computacional considerable. Otra desventaja es que: sería inútil haber generado parte de las claves si es que una de ellas ya encuentra el texto claro correcto.

Ya que este algoritmo se ha construido para evaluar 1023 claves, tal vez la diferencia no sea apreciable, pero si el número de bits que conforman la clave aumentan en tamaño, como en el caso del algoritmo DES cuya longitud de clave es 56 bis, el hecho de generar una matriz con todas las claves ocupará un tiempo considerable, causando esto una pérdida de eficiencia.

Para evitar este problema, en esta opción se genera una clave e inmediatamente se descifra y a continuación se verificar si es la correcta o no, utilizando el filtrado obtenido en la sección anterior.

En la tabla 2.1 se presentan valores reales obtenidos generando una matriz de posibilidades, el tiempo aumenta de manera notoria, llegando hasta los 40 minutos aproximadamente cuando la longitud es 25 bits, siendo evidente el costo de procesamiento, por este motivo se propone el ahorro de esta matriz, otra desventaja cuando la longitud de la clave aumenta es que si supera los 25 bits para el caso de una computadora con recursos promedio, no puede realizar dicha tarea dado que Matlab valora los recursos existentes y limita el tamaño de una matriz, como solución se presenta la computación paralela.

Tabla 2.1. Tiempos de acuerdo a longitud de clave.

	Posibilidades	**Tiempo**
x=10	1023	0.381 s
x=15	32767	2.21 s
x=20	1048575	70 s
x=25	33554431	2473,14 s

Para comprender el funcionamiento, se realiza un ejemplo que sigue los diagramas de flujo.

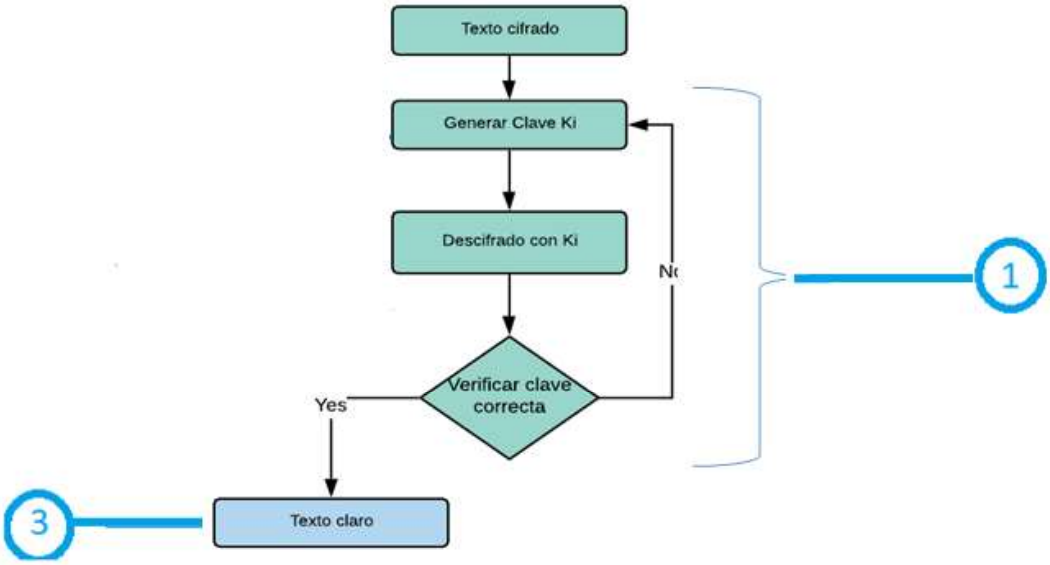


Figura 2.37. Diagrama de Flujo de Ataque a S-DES sin matriz de claves

Instrucciones
Paso 1(figura 2.37) Para iniciar se necesita un texto cifrado con lo cual se genera la primera clave posible (figura 2.38) que no será almacenada y se aplica la función de descifrado dentro de un cliclo for. Dado el texto cifrado: *"M«LóÆñèÞîÂÆÚ‡ÚðLfiL©ì«ìÀñ«ÚLÚÞÚ-œìL©ìL‡œìÆKÚL¶ÆœÞÚLÚL…ÚL‡îÆÀÚL©ìLÆìóœèìÆÚÆLœ«ÚLó…Ú÷ìLèÆî¶Ú«©îLÞî©ÚfL…ÚfLóîÀ¶ñ«Úóñî«ìfLèîfñ¶…ìfL´ÚfÞÚLì«óî«ÞÆÚÆLÚ-œì……ÚL-œìLèìÆÀñÞìLì…LÚóóìfîðLì…LÞìAÞîLèœì©ìLÞì«ìÆL«¡ÀìÆîfÁ}3T„L$Lf À¶î…îfÁh#-„Ù"* **Clave** I_1 I_2 I_3 I_4 I_5 I_6 I_7 I_8 I_9 I_{10} 0 0 0 0 0 0 0 0 0 1 **Figura 2.38.** Clave ki Dentro de un ciclo for limitado por 1023, que será la cantidad máxima que se descifre con un clave posible, se llama a la función key, la cual toma como argumento de entrada la primera posibilidad de clave, 10 bits con todos sus valores en cero y el tipo, siendo 1 para la sub-clave 1 y 2 para sub-clave 2.

Posteriormente se descifra, utilizando k1 y k2, tomando los 80 primeros caracteres del texto cifrado, almacenado esto en un vector clave 1.

La verificación del descifrado correcto se da utilizando los criterios antes mencionados, en el caso de que no cumpla con la condición se procede a genera una nueva clave, como es en este caso, ya que el vector “solu” contiene un conjunto de caracteres sin sentido en el lenguaje español.
De modo que como indica el diagrama de flujo, el ciclo for se repetirá generando una nueva clave, vector clave 2.
El proceso es repetitivo, así que se descifra y verifica obteniendo nuevamente una negativa, lo que conlleva a generar otra clave.
En el presente ejemplo se probó una a una las 7 primeras claves dando como resultado un no, en la condición de verificación; sin embargo, con la octava clave se obtuvo un éxito en el ataque.
En la figura 2.39 se muestran algunas de las claves obtenidas, haciendo énfasis en la clave correcta.

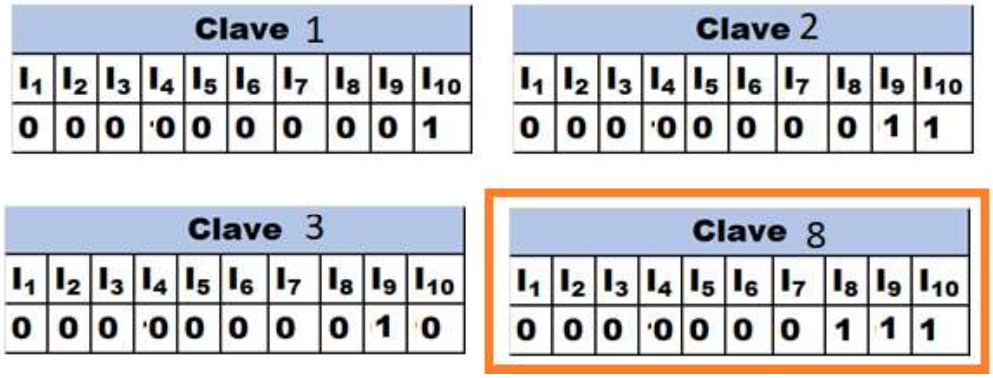


Figura 2.39. Claves evaluadas.

Código de MATLAB:

```
cifrado="M«LóÆñèÞîÂÆÚ‡ÚðLƒìL©ì«îÀñ«ÚLÚÞÚ–œìL©ìL‡œìÆKÚL¶ÆœÞÚLÚL…Ú
L‡îÆÀÚL©ìLÆìóœèìÆÚÆLœ«ÚLó…Ú÷ìLèÆî¶Ú«©îLÞî©ÚƒL…ÚƒLóîÀ¶ñ«ÚóñîìƒLè
îƒñ¶…ìƒL´ÚƒÞÚLì«óî«ÞÆÚÆLÚ–œì……ÚL–œìLèìÆÀñÞìLì…LÚóóìƒîðLì…LÞìAÞîL
èœì©ìLÞì«ìÆL«¡ÀìÆîƒÁ}3T„L$Lƒ À¶î…îƒÁh#–„Ù”

for z=1: size(1023,1)
K1=keys(fliplr(dec2binvec(z,10)),1);
K2=keys(fliplr(dec2binvec(z,10)),2);
```

```
solu=descifrarSerial(cifrado,K1,K2);
count=(length(find((solu>=123 & solu<=255)|(solu>=1&
solu<=31)|(solu>=33& solu<=43))))/length(cifrado);

count1=(length(find(solu==97|solu==101|solu==105|solu==111|solu=
=117)))/length(cifrado);

    if count*100<10 && count1*100>25
       break;

           z;
           solu;
    end
end
```

Paso 2(figura 2.37)

El valor descifrado obtenido fue:

"En criptografía, se denomina ataque de fuerza bruta a la forma de recuperar una clave probando todas las combinaciones posibles hasta encontrar aquella que permite el acceso, el texto puede tener números (1234) y símbolos (·#).”

Código de MATLAB:

```
           solu;
```

Los resultados obtenidos en el ataque en forma serial por los 2 métodos, se resumen en la tabla 2.2.

Tabla 2.2. Tiempos de ataque con matriz y sin matriz de claves

Texto Corto		Texto Largo	
Serial	**Tiempo(s)**	**Serial**	**Tiempo(s)**
Con Matriz Claves	78.6	Con Matriz Claves	79,66
Sin Matriz Claves	73.2	Sin Matriz Claves	76,45

2.5. PARALELIZAR EL ALGORITMO DE ATAQUE

Se desea transformar el algoritmo creado con el afán de hacerlo más eficiente respecto al tiempo, para lo cual se consideran 2 opciones:

- ✓ Reducción de espacio de análisis del texto cifrado
- ✓ Paralelizar con *toolbox* PTC

2.5.1. REDUCCIÓN DE ESPACIO DE ANÁLISIS DEL TEXTO CIFRADO

Tomando en cuenta que el objetivo del ataque es encontrar el texto en claro y que el total contenido se cifra con una misma clave se ha decidido utilizar únicamente una sección del texto para el proceso y una vez encontrada la clave proceder a realizar el descifrado completo obteniendo el resultado esperado.

Se establece el tamaño del segmento de 80 caracteres, este valor tomado por el número de caracteres con el que se llena una línea con los márgenes comunes en Microsoft Word, así que el tiempo se reducirá notablemente.

Si el texto tiene una longitud menor a 80, se procesa todo el texto, si no se tomará los 80 primeros caracteres.

2.5.2. PARALELIZAR CON TOOLBOX PTC

Dentro del *Parallel Computing Toolbox* existen diversos comandos que permiten la paralelización de un código serial, a continuación, se paraleliza el algoritmo utilizando:

-PARFOR

-SPMD

- ✓ **Paralelizar algoritmo con PARFOR**

Se procede analizar el código de ataque para evaluar donde se podrá optimizar, como se conoce, el algoritmo contiene 4 ciclos for:

1. Genera matriz de posibilidades de claves
2. Evalúa cada posibilidad, bucle externo.
3. Descifra hasta 80 caracteres.
4. Descifrar hasta la longitud del cifrado.

Es necesario comparar tiempos de procesamiento, for vs parfor. Se considera para dicho proceso un texto de 3 páginas con clave 1111111111, se realizan 3 ejecuciones ya que los

tiempos varían dependiendo de factores internos de Matlab. El clúster local se configura con 4 workers únicamente para evaluar el paralelismo.

Primer Lazo

Se remplaza el primer ciclo por un parfor, no es necesario cambiar nada en el código ya que las variables que pertenecen a este bucle están dentro de la clasificación antes mencionada, el contador es entero y no existe recursividad.

Tabla 2.3. Tiempos de ejecución FOR vs PARFOR en lazo 1.

Lazo 1			
Sentencia	**Ejecución 1**	**Ejecución 2**	**Ejecución 3**
For	0.16	0.08	0.08
Parfor	0.14	0.13	0.11

Para identificar cuando se obtiene mayor eficiencia al crear la matriz de posibilidades de claves se crea la tabla 2.3 donde se observa que el uso de Parfor no es eficiente por lo que lo correcto sería dejar la sentencia for en el lazo, otra opción sería utilizar la vectorización para generar la matriz de claves, intercambiando el lazo for, por la siguiente línea de código: `t=1:1:1023;`

La tabla 2.4 muestra la diferencia de tiempos al utilizar vectorización, de hecho, el tiempo es menor por lo que se decide utilizar esta forma para generar la matriz.

Tabla 2.4. Tiempos de ejecución FOR, PARFOR vs VECTORIZACIÓN en lazo 1.

Lazo 1			
Sentencia	**Ejecución 1**	**Ejecución 2**	**Ejecución 3**
For	0.16	0.08	0.08
Parfor	0.14	0.13	0.11
Vectorización	0.02	0.03	0.03

Segundo lazo

Lazo externo, que evalúa cada posibilidad de clave existente, descifra y encuentra la clave con la que se cifró, por ende, contiene 2 lazos que descifran (Tercer y cuarto lazo). Se toman los tiempos del lazo cuando se utiliza for y parfor (Los bucles internos se mantiene en for).

Tabla 2.5. Tiempos de ejecución FOR, PARFOR en lazo 2.

Lazo 2			
Sentencia	**Ejecución 1**	**Ejecución 2**	**Ejecución 3**
For	76.20	74.83	78.27
Parfor	38.56	37.38	43.72

Tercer y cuarto lazo

En el caso de 3 y 4, se obtienen dos errores:

1. Error de anidado, ya que estos bucles están dentro de otro parfor, así que se construye una función de nombre “descifrar”, donde se remplaza el for por un parfor, esta será llamada en el cuerpo del bucle principal en 2 ocasiones.
2. Un error en la sentencia break cuando se ejecuta el bucle parfor, por lo que habrá que eliminar esta sentencia. Dentro de un ciclo paralelizado no puede existir un break ni return, debido a que los trabajadores funcionan de manera independiente así que, si uno llegó hasta esa instancia y otro no, se puede dañar su trabajo al intentar romper dicho bucle.

Tabla 2.6. Tiempos de ejecución FOR, PARFOR en lazo 3.

Lazo 3			
Sentencia	**Ejecución 1**	**Ejecución 2**	**Ejecución 3**
For	0.08	0.09	0.10
Parfor	0.14	0.14	0.14

Como muestra la Tabla 2.5, el for es más eficiente ya que descifra únicamente 80 caracteres.

Tabla 2.7. Tiempos de ejecución FOR, PARFOR en lazo 4.

Lazo 4			
Sentencia	**Ejecución 1**	**Ejecución 2**	**Ejecución 3**
For	6.88	6.72	10.35
Parfor	2.94	2.93	2.99

En este la Tabla 2.6. muestra que si bien este lazo realiza el mismo proceso de descifrado (lazo 3), el parfor es mejor ya que el tiempo que se ahorra es superior al 50% del obtenido con el lazo for.

Se concluye que, para optimizar el código del ataque, se utilizará vectorización para generar matriz de claves, en el lazo 2 y 4 se utilizará parfor. Con estos cambios se ejecuta el algoritmo total para ver resultados de eficiencia.

Se puede observar el algoritmo paralelo en el Segmento de código 2.6, además tomando en cuenta los parámetros analizados, se procedió a paralelizar el cifrado y descifrado esto detallado en el Segmento de código 2.7 y 2.8 respectivamente.

```
function ataque(cifrado,ParteCifrado)

p=2^(10);
t=1:1:1023;

j=0;
te=0;
parfor z=1:p-1

    K1=keys(dec2binvec(t(z),10),1);
    K2=keys(dec2binvec(t(z),10),2);
   solu=descifrarParalelo(ParteCifrado,K1,K2);
     count=(length(find((solu>=123 & solu<=255)|(solu>=1&
solu<=31)|(solu>=33& solu<=43))))/length(ParteCifrado);

count1=(length(find(solu==97|solu==101|solu==105|solu==111|
solu==117)))/length(ParteCifrado);

    if count*100<10 && count1*100>25
              j=j+z;
        if length(cifrado)<80
             te=te+solu;

        else

           solu=descifrarParalelo(cifrado,K1,K2);
           te=te+solu;
        end
    end

end

llave=j;

txt=char(te);
 save auxiliar.mat llave txt
end
```

Segmento de código 2.6. Algoritmo de ataque al S-DES

```
function [cifrado]=cifrarParalelo(texto,K1,K2)
valor = double (texto);

parfor hh=1:length(valor)
textoVector = dec2binvec (valor(hh),8);
IP=[textoVector(2)       textoVector(6)       textoVector(3)
textoVector(1) ...
textoVector(4)         textoVector(8)         textoVector(5)
textoVector(7)];
R=[IP(5) IP(6) IP(7) IP(8)];
Fk1=fk(IP,K1);
SW=[R Fk1];
R2=[SW(5) SW(6) SW(7) SW(8)];
Fk2=fk(SW,K2);
aux=[Fk2 R2];
PIinversa= [aux(4) aux(1) aux(3) aux(5) aux(7) aux(2) aux(8)
aux(6)];
pp1=bi2de(PIinversa);
cifrado(hh)=char(pp1);
end
cifrado
end
```

Segmento de código 2.7. Cifrado S-DES paralelo.

```
function [solu]=descifrarParalelo(cifrado,K1,K2)
parfor hh=1:length(cifrado)
            textoVector_d = dec2binvec
(double(cifrado(hh)),8);
            IP=[textoVector_d(2) textoVector_d(6)
textoVector_d(3) textoVector_d(1) ...
            textoVector_d(4) textoVector_d(8)
textoVector_d(5) textoVector_d(7)];
            IP_Dcha=[IP(5) IP(6) IP(7) IP(8)];
            Fk2=fk(IP,K2);
            SW=[IP_Dcha Fk2];
            Fk1=fk(SW,K1);
            IP_Dcha_2=[SW(5) SW(6) SW(7) SW(8)];
            aux=[Fk1 IP_Dcha_2];
            PIinversa= [aux(4) aux(1) aux(3) aux(5) aux(7)
aux(2) aux(8) aux(6)];
            aux=bi2de(PIinversa);
```

Segmento de código 2.8. Descifrado S-DES paralelo

- ✓ **Paralelizar con SPMD**

Conocida la funcionalidad de SPMD, se aplica al algoritmo de fuerza bruta para romper al S-DES, algo importante para tomar en cuenta es que a diferencia de PARFOR, es que, si se utiliza condicionales para separar trabajo entre laboratorios, es necesario fijar un número de workers, porque si se condiciona el trabajo para 5 laboratorios y el parallelPool inicia con 2, 3 o 4, ocurrirá un error en la ejecución.

Ahora se realizará un cambio en el código serial, de tal manera que el trabajo se divida en 2 workers.

Se inicia con la creación de un objeto compuesto, para lab 1, se asignan las claves desde 0000000001 hasta 0111111111, mientras que para lab 2 se asignan claves desde 1000000000 hasta 1111111111(Segmento de código 2.9). En la figura 2.40 se visualiza como se forman las matrices dentro del objeto creado.

```
x=Composite();
%Cada índice indica el
labindex de cada trabajador
%Se divide la matriz de
claves en 2 secciones
x{1}=GeneLlaves
(1:1:512,:);
x{2}=GeneLlaves
(513:1:1024,:);
```

Segmento de código 2.9. División de matriz de claves con COMPOSITE.

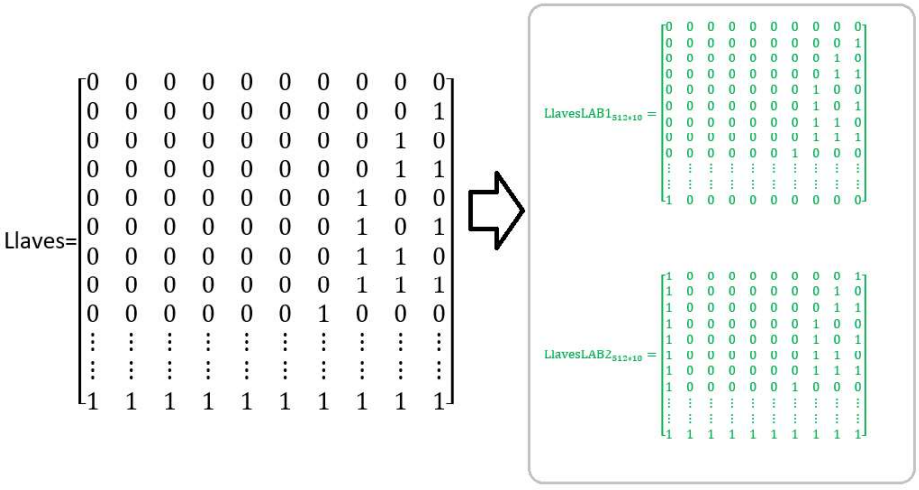


Figura 2.40. Asignación de posibilidades para LAB 1 y 2.

Ahora se elige utilizar SWITCH-CASE(segmento de código 2.10), para acceder a cada trabajador, con su respectiva sección de la matriz de claves.

```
spmd
    %Un trabajador ejecuta la función establecida
    %según sea el caso, de acuerdo a labindex.
    switch labindex
        %Trabajador 1
        case 1
            %Función que evalua la sección de la
            %matriz de llaves, que corresponde a lab1
            AtaqueSDES_Serial_I(cifrado,x);

        case 2
            %Función que evalua la sección de la
            %matriz de llaves, que corresponde a lab2 y
            AtaqueSDES_Serial_I(cifrado,x);

    end
```

Segmento de código 2.10. Ataque por fuerza bruta a S-DES con SPMD.

2.6. IMPLEMENTACIÓN DE RED DE PRUEBAS

Como se comentó al inicio, con el fin de probar este algoritmo, se implementará dos tipos de red: una cableada y una inalámbrica (figura 2.41).

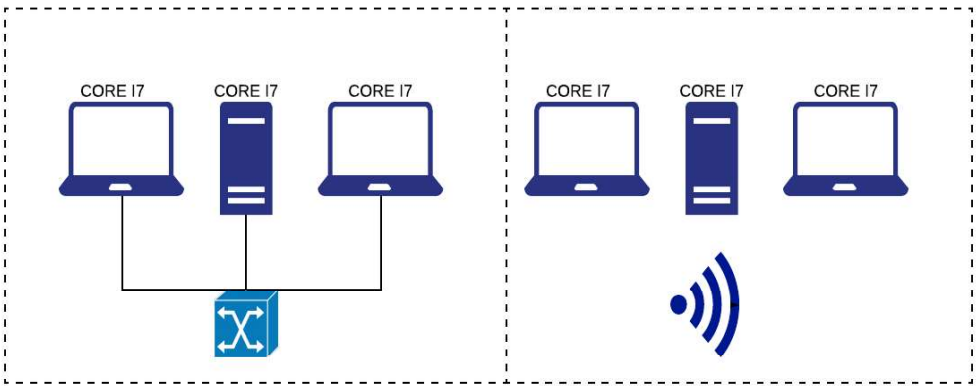


Figura 2.41. Esquema de redes a implementar LAN (izquierda) y ADHOC (derecha).

a) Implementación de red cableada

Con el objetivo de construir un clúster, se procede a la creación de una red local que estará conformada por 3 computadoras y un dispositivo de conmutación. Información detallada en la Tabla 2.8.

Se comprueba la conectividad, utilizando el comando ping, entre todos los equipos, una vez obtenido éxito en el proceso se selecciona el host que será el cliente: host 2.

Es indispensable que el firewall de Windows esté desactivado en cada uno de los elementos de la red, para que no existan problemas futuros como bloqueo de puertos que se utilizarán en la instalación de los servicios necesarios para crear el clúster. Además, es recomendable desactivar el antivirus para evitar situaciones similares a las mencionadas anteriormente.

Una vez conectado correctamente se realiza un direccionamiento como lo muestra la tabla 2.9.

Tabla 2.8. Información de Equipos de Red LAN.

Equipos	Detalle
ASUS ViviBook Flip 14	PROCESADOR Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz (4 CPUs), ~2.9GHz RAM Memory: 8192MB RAM
TOSHIBA Satélite Radius P55W –B	PROCESADOR Intel(R) Core(TM) i7-4510uçU CPU @ 2.00GHz (4 CPUs), ~2.6GHz RAM Memory: 8192MB RAM
ADIKTO	PROCESADOR Intel(R) Core(TM) i7-7700U CPU @ 3.60GHz (8 CPUs), ~3.6GHz RAM Memory: 16384MB RAM Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz (4 CPUs), ~2.9GHz RAM Memory: 8192MB RAM
Switch D-LINK	No administrable

Tabla 2.9. Direccionamiento Red LAN.

Nombre de host	Dirección IP
HOST1: Vane	192.168.2.7
HOST2: TANNIA-PC	192.168.2.6
HOST 3: DESKTOP-544UI95	192.168.2.8

b) Implementación de una red AD-HOC

Para conectar las 3 computadoras inalámbricamente es necesario la creación de una red Ad-Hoc a la cual se conectarán los demás equipos para obtener una conectividad satisfactoria.

En la tabla 2.10 se puede observar los hosts que participan en esta conexión con sus respectivas direcciones IP modificadas.

Tabla 2.10. Direccionamiento Red AD-HOC.

Nombre de Host	Dirección IP
HOST1: Vane	192.168.2.9
HOST2: TANNIA-PC	192.168.2.10
HOST 3: DESKTOP-544UI95	192.168.2.11

Requisito: En las opciones avanzadas del centro de recursos y redes compartidas, ACTIVAR la opción de descubrimiento de la red. Figura 2.42.

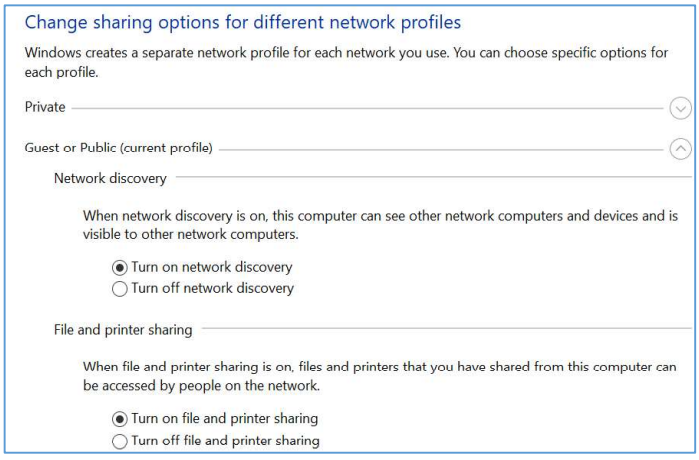


Figura 2.42. Configuración de perfiles de red compartida.

Es necesario comprobar que el equipo seleccionado pueda albergar una red Ad-hoc, para lo cual desde el CMD se digita el siguiente comando: *netsh wlan show drivers,* obteniendo un resultado similar al que muestra la figura 2.43, donde se confirma que este equipo admite la creación de una red hospedada, este proceso se configura en el host3.

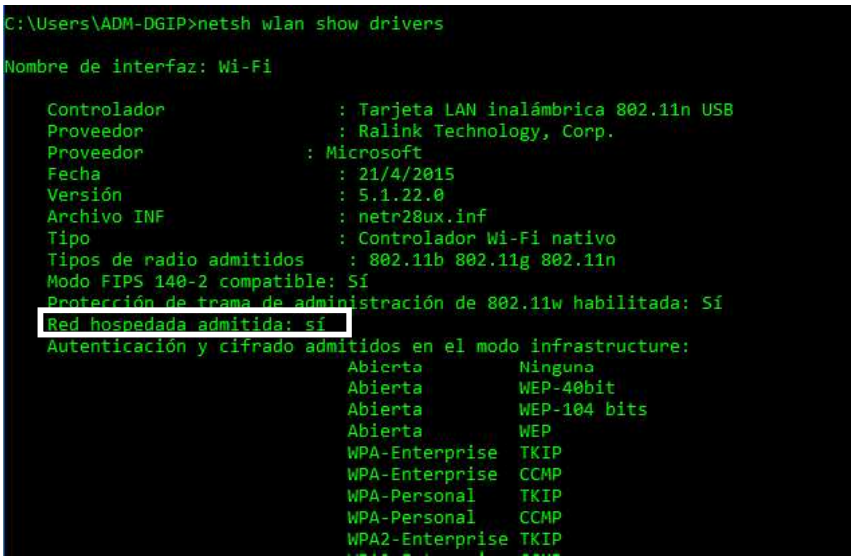


Figura 2.43. Admisión de red AD-HOC.

Para crear la red, se utiliza el siguiente comando:

netsh wlan set hostednetwork mode=allow ssid=NombreRed key=contraseña

Donde se establece el nombre y la contraseña con la que se permitirá el acceso a la red, en la figura 2.44 se observan los valores con los que se creó la red.

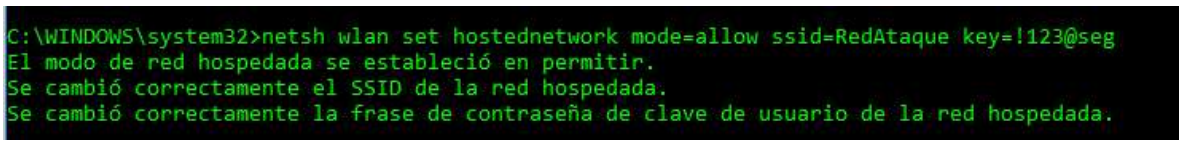


Figura 2.44 Creación de red AD-HOC.

Una vez creada la red, se procede activarla, obteniendo un mensaje igual al de la figura 2.45 si todo ha salido bien.

Figura 2.45. Inicio de red AD-HOC.

En el caso de inutilizar esta red, el adaptador creado se deshabilita con el comando que muestra la figura 2.46.

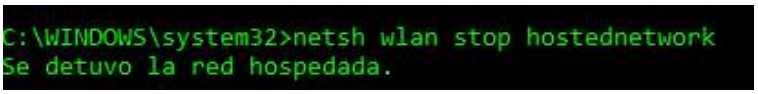


Figura 2.46. Finalización de red AD-HOC.

Se procede a establecer las direcciones IP inicialmente descritas para cada host, en las propiedades del adaptador de red virtual que se ha creado con los pasos anteriores.

Ahora se procede a conectar los 2 equipos restantes por medio de WI-FI con las credenciales establecidas y configurar las direcciones IP restantes que se indican en la tabla 2.9.

Se comprueba la conectividad desde el CMD, con el comando: ping dirección IP destino, si la respuesta del comando es satisfactoria entonces la red se encuentra configurada.

2.7. ATAQUE EN FORMA PARALELA

Como se mencionó anteriormente el costo computacional para la ejecución en serie de un conjunto de líneas de código dependiendo de sus características puede ser alto. Para resolver este problema se lo debe ejecutar paralelamente.

Esta aplicación se ejecutará:

- Con un solo PC
- Con varios PCs

2.7.1. CON UN SOLO PC

Matlab permite, mediante el uso del PCT descrita en el capítulo anterior la paralelización de una aplicación en un mismo equipo, conocido en como clúster local.

Se considera el código construido y ejecutado en la sección anterior, al analizar su funcionamiento se puede decir que los ciclos for, generan mayor tiempo de procesamiento por el número de vueltas que se establecen por límite.

2.7.1.1. Creación de un clúster local

Para ejecutar un script de manera paralela local no es necesario crear un clúster local, ya que Matlab lo tiene por defecto, para acceder a este existen 2 maneras de iniciar Matlab Parallel Pool:

a) Mediante la interfaz gráfica:

Se debe ingresar a la pestaña Home, ir a la opción parallel, en clúster por defecto aparecerá marcada la opción local (figura 2.47), también existe Matlab Parallel Cloud (permite ejecutar su aplicación en trabajadores MATLAB que se ejecutan en clústeres personalizables con tecnología de Amazon EC2. Cuando su clúster en la nube se está ejecutando, accede a él de la misma manera que accede a un clúster en su propia red en el sitio).

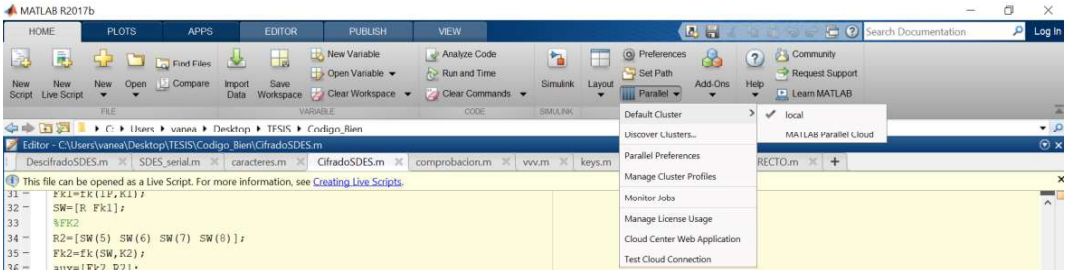


Figura 2.47. Elección de clúster local.

En el extremo inferior izquierdo de Matlab, se encuentra un ícono como el siguiente:  , al presionarlo se desplegarán 2 opciones como se muestra en Figura 2.48, preferencias o iniciar parallel pool.

Figura 2.48. Opciones del parallel pool.

En las preferencias del *toolbox*, se encuentran detalles del clúster, como ya se eligió será el local, para la construcción del *parallel pool*, se pueden establecer un número de *workers* a utilizar dentro del grupo, este no puede ser superior al número de trabajadores del clúster.

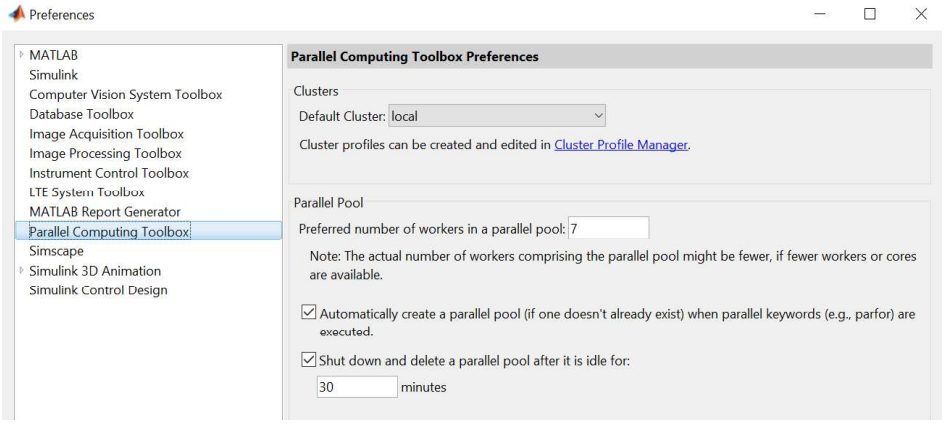


Figura 2.49. Preferencias de *Parallel Computing Toolbox*

Si se desea cambiar las propiedades del clúster, se ingresa a Clúster Profile Manager, donde se visualiza una ventana como indica la figura 2.49.

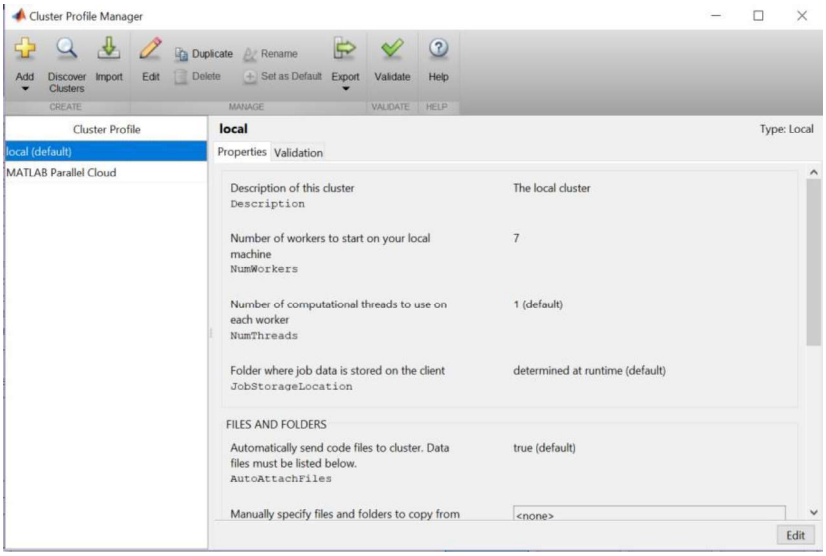


Figura 2.50. Propiedades del Clúster local

Los clústeres se crean mediante un perfil, el mismo que contiene los parámetros que lo componen, así que a continuación se modificarán características del perfil del clúster.

El valor principal en este caso, será el número de trabajadores, cuyo valor suele ser igual al número de procesadores físicos del equipo, para realizar pruebas se ha cambiado a 7, para esto se selecciona el botón “Edit” y se cambia el valor (figura 2.50).

Se procede a validar el perfil(opcional), esto asegura que la sesión del cliente de MATLAB puede acceder al clúster y que el clúster puede ejecutar los diversos tipos de trabajos con la configuración de su perfil.

Las etapas de la validación de perfil incluyen cinco pasos:

- Se conecta al clúster
- Ejecuta un trabajo independiente en el clúster utilizando el perfil
- Ejecuta un trabajo de comunicación de tipo SPMD en el clúster utilizando el perfil
- Ejecuta un trabajo de comunicación de tipo agrupación en el clúster utilizando el perfil
- Ejecuta un trabajo de grupo paralelo en el clúster utilizando el perfil

Cuando se completen las pruebas, puede seleccionar Mostrar informe para obtener más información sobre los resultados de las pruebas. Esta información incluye los mensajes de error, los registros de depuración y otros datos que pueden ser útiles para diagnosticar problemas o ayudar a determinar la configuración de red adecuada (figura 2.51)

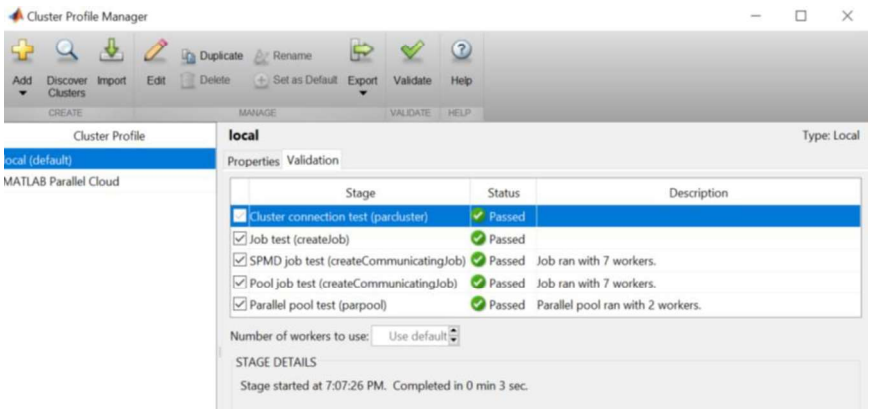


Figura 2.51. Validación del Clúster local.

Una vez configuradas las preferencias, se procede a inciar el clúster, el cual se queda activo por 30 minutos si no se cambió en el paso anterior.

Se obtiene un mensaje:

Starting parallel pool (parpool) using the 'local' profile ...

Esto indica que el clúster local, está listo para ejecutar cualquier script.

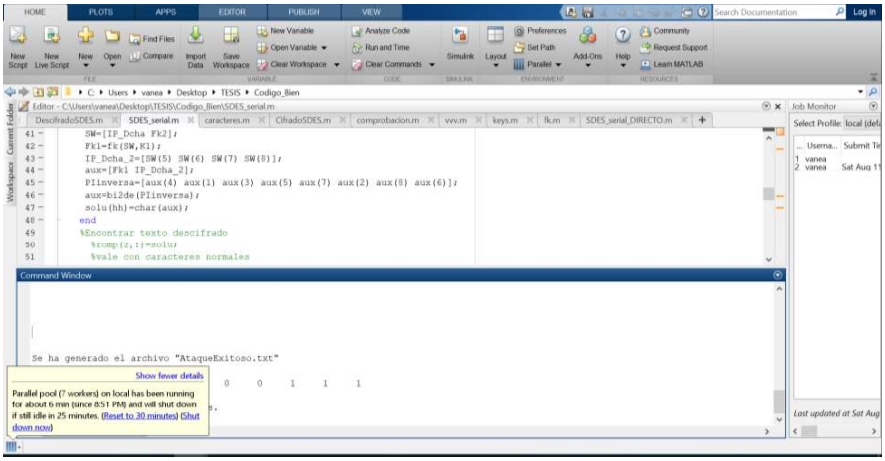


Figura 2.52. Inicio del parallel pool.

b) Mediante comandos

Para crear el Parallel Pool, es necesario instanciar el clúster a utilizar, en esta ocasión el clúster lo conforma una sola computadora con sus procesadores, este establecido por defecto como 'local', a través del objeto c de tipo clúster se puede acceder a los parámetros y características del mismo mediante el uso del punto "."(figura 2.52), se establecerá un número de workers cualquiera tomando en cuenta los procesadores, Matlab aconseja que el número de workes máximo será el valor de los procesadores físicos del ordenador; sin embargo, este valor puede variar hasta 12, sin que se asegure la eficiencia en la ejecución, para que el valor se guarde en el perfil, es necesario utilizar la función saveProfile donde el argumento de entrada será el clúster.

Ahora se instancia el ParPool, mediante un constructor que solicita como argumento de entrada el clúster previamente creado. Posteriormente se codifica lo requerido y se finaliza el ParPool, esta línea puede obviarse si el tiempo necesario para la aplicación sea menor a 30 minutos, valor que se establece por defecto para "apagar" el grupo. Segmento de código 2.11.

La diferencia del método por la interfaz gráfica es que, se elige los workers del clúster que se utilizarán en la ejecución de un script, mientras que en este método se utiliza la totalidad de *workers* del clúster.

```
myCluster = parcluster('local');
myCluster.NumWorkers = 7;
saveProfile(myCluster);
pool1=parpool(myCluster);
%Proceso a realizar
%en paralelo
%..........
delete(myCluster);
```

Segmento de código 2.11. Creación del paralell poll con el clúster local.

2.7.2. CON VARIOS PCS

Para la utilización de varios computadores en una ejecución es necesario formar un clúster.

2.7.2.1. Creación y configuración de un clúster multi-computador

Se utilizará el MDCS para la construcción de un clúster con la asociación de equipos conformada.

a. Instalación del MDCS

Se puede realizar la instalación, creación de *JobManager* y *Workers* desde el CMD o mediante la interfaz gráfica.

- ✓ ***Instalación desde el CMD***

Desde el CMD, se ingresa al directorio C:\Program Files\MATLAB\R2017b\toolbox\distcomp\bin, donde se procede a ingresar el comando *mdce install*, si el comando se procesa correctamente el mensaje obtenido es MATLAB Distributed Computing Server installed cómo se muestra en la figura 2.53. Ahora se procede a levantar el servicio previamente instalado mediante el comando *mdce start*, se obtiene el mensaje *MATLAB Distributed Computing Server started.* Este procedimiento se realiza en todos los equipos.

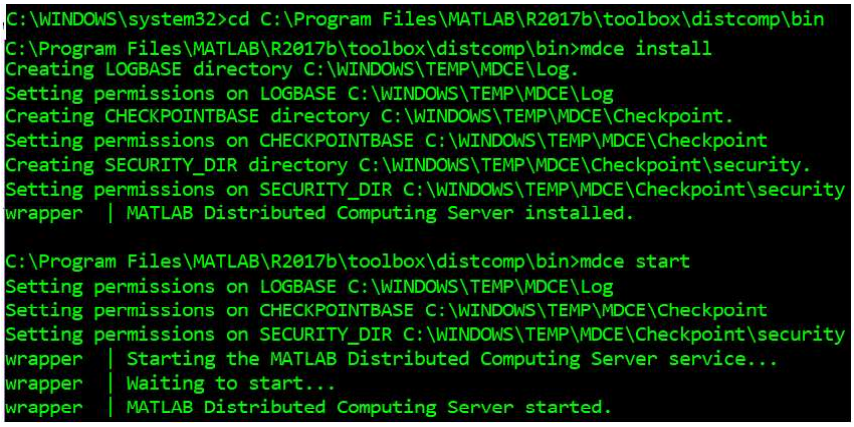


Figura 2.53. Instalación del servicio MDCE.

En el caso que se presente un problema durante la instalación o inicialización del servicio, se utiliza *mdce uninstall -clean* para empezar la configuración otra vez.

Job manager

Se procede a crear el Job Manager (únicamente en el cliente), este distribuye las actividades a realizarse entre los trabajadores, el nombre ingresado será el que identifique al clúster.

startjobmanager.bat -name nombre_clúster -v

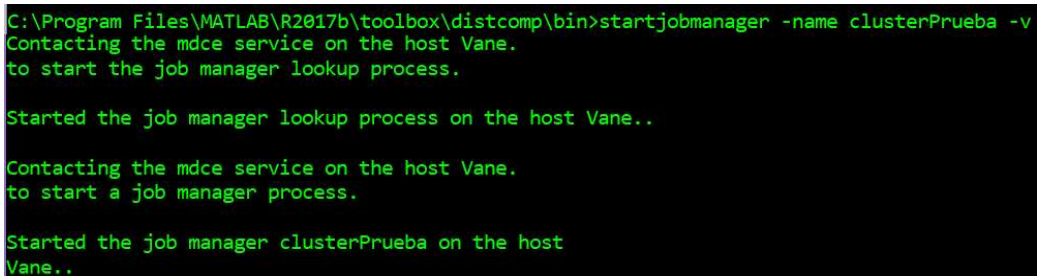


Figura 2.54. Inicio de JobManager.

En la figura 2.54 se muestra el resultado del correcto funcionamiento del comando ingresado.

Creación de workers mediante comandos

Para la creación de los trabajadores locales, se utiliza el siguiente comando:

Startworker -jobmanagerhost nombre_host –jobmanager nombre_clúster -name nombre_worker

Donde se especifica el nombre del host, el nombre del clúster y finalmente el nombre del trabajador (figura 2.55).

```
C:\Program Files\MATLAB\R2017b\toolbox\distcomp\bin>startworker -jobmanagerhost Vane
-jobmanager clusterPrueba -name host1Worker2
```

Figura 2.55. Creación de un worker local.

Desde el cliente se pueden crear los trabajadores de los demás equipos, Workers Remotos, utilizando el siguiente comando*:*

startworker.bat -jobmanagerhost nombre_host_local -jobmanager nombre_clúster -remotehost nombre_host_remoto -name nombre_worker.

Donde se especifica el nombre del host, el nombre del clúster, nombre del host remoto y finalmente el nombre del trabajador (figura 2.56).

```
C:\Program Files\MATLAB\R2017b\toolbox\distcomp\bin>startworker -jobmanagerhost Vane
-jobmanager ClusterPrueba -remotehost Tannia-PC -name HOSTREMOTO1
```

Figura 2.56. Creación de un *worker* remoto

Para comprobar la creación correcta de *workers* dentro de los hosts especificados se ingresa en el CMD el comando: *nodestatus -infolevel3,* este proporciona un resumen de la configuración total realizada(figura 2.57).

Algunos de los datos presentes en la salida del comando descrito son:

- Información esencial del MDCE service como versión, plataforma, nivel de seguridad, etc
- El estado del Job Manager
- Datos del host, clúster y trabajadores.
- Tiempo
- Tamaño en bytes

```
C:\Program Files\MATLAB\R2017b\toolbox\distcomp\bin>nodestatus -infolevel 3
MDCE service:
     Security level                0
     Version                       6.11
     MATLAB                        C:\Program Files\MATLAB\R2017b
     mdce_def File                 C:\Program Files\MATLAB\R2017b\toolbox\distcomp\bin\mdce_def.bat
     Platform                      win64

Job manager lookup processes:
     Status                        Running

Job manager:
     Name                          ClusterPrueba
     Running on host               Vane.
     Number of workers             4
     Status                        running
     Supported releases            R2017b
     Worker names and host names   Host1worker1, Vane.
                                   Host1worker2, Vane.
                                   Host1worker3, Vane.
                                   Host1worker4, Vane.
     Start time                    Mon Jul 23 20:23:14 COT 2018
     Port                          [27357, 27356]
     Requested job manager lookup
        processes                  Vane.:27350
     Registered with job manager
        lookup processes on hosts  Vane.:27350
     Database size in bytes        3156071
     VM heap size in bytes         128974848
     Database item cache size      0
```

Figura 2.57. Salida del comando nodestatus -infolevel 3 parte I.

La salida del comando es bastante extensa, ya que la información es detallada de host en host y de *worker* en *worker*, la figura 2.58 muestra un fragmento que corresponde al primer host, que describen sus 2 primeros *workers*.

```
Worker:
     Name                          Host1worker1
     Running on host               Vane.
     Status                        Idle
     Job manager                   ClusterPrueba
     Connection with job manager   Connected
     Job manager hostname          Vane.
     Start time                    Mon Jul 23 20:24:09 COT 2018
     Port                          [27358]
     Requested job manager lookup
        processes                  Vane:27350
     Registered with job manager
        lookup processes on hosts  Vane.:27350
     AttachedFiles folder          C:\WINDOWS\TEMP\MDCE\Checkpoint\Vane._Host1worker1_mlworker_log\matlab\filedepe
ndencies
     Worker startup directory      C:\WINDOWS\TEMP\MDCE\Checkpoint\Vane._Host1worker1_mlworker_log\matlab\work
     Network addresses of host     2001:0:5ef5:79fd:cd5:1594:417d:2b58
                                   192.168.0.3
                                   fe80:0:0:0:cd5:1594:417d:2b58

Worker:
     Name                          Host1worker2
     Running on host               Vane.
     Status                        Idle
     Job manager                   ClusterPrueba
     Connection with job manager   Connected
     Job manager hostname          Vane.
     Start time                    Mon Jul 23 20:25:27 COT 2018
     Port                          [27359]
     Requested job manager lookup
        processes                  Vane:27350
     Registered with job manager
        lookup processes on hosts  Vane.:27350
     AttachedFiles folder          C:\WINDOWS\TEMP\MDCE\Checkpoint\Vane._Host1worker2_mlworker_log\matlab\filedepe
ndencies
     Worker startup directory      C:\WINDOWS\TEMP\MDCE\Checkpoint\Vane._Host1worker2_mlworker_log\matlab\work
     Network addresses of host     2001:0:5ef5:79fd:cd5:1594:417d:2b58
                                   192.168.0.3
                                   fe80:0:0:0:cd5:1594:417d:2b58
```

Figura 2.58. Salida del comando nodestatus -infolevel 3 parte II.

✓ **Utilización de la interfaz gráfica**

Admincenter es un entorno gráfico que permite administrar *hosts*, *schedulers* y *workers*, por lo que este puede suplir a ciertos comandos que fueron detallados en la sección anterior. Para poder utilizar esta herramienta es necesario que el MDCE esté previamente instalado en todos los elementos de la red, desde el CMD se ingresa el comando: admincenter, este desplegará una nueva ventada con diferentes opciones de administración para el clúster, como se muestra en la figura 2.59.

Figura 2.59. Inicio del centro de administración.

El centro de administración, se dividen en 3 secciones como lo indica la figura 2.60, la primera se refiere a hosts, la segunda al Scheduler y la tercera a los trabajadores.

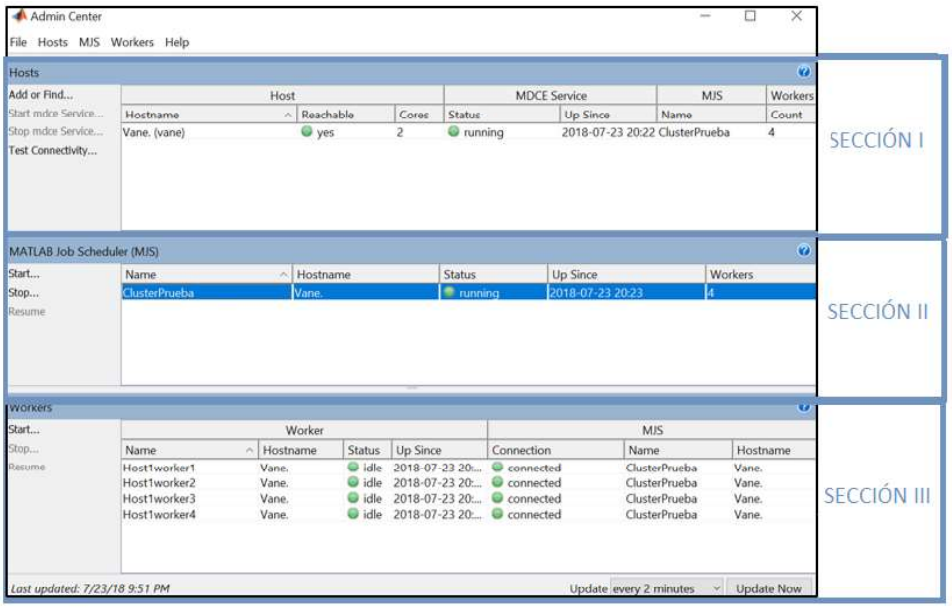


Figura 2.60. Centro de Administración.

En la primera sección se procede a añadir los hosts al clúster, presionando la opción *Add or Find,* con lo se despliega una ventada como la que se muestra en la Figura 2.61, se especifica el rango de la red, para que reconozca los ordenadores existentes.

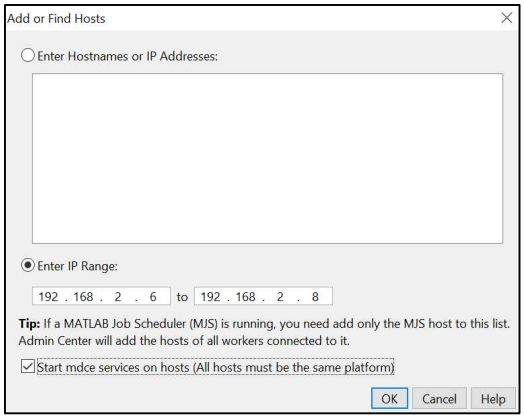


Figura 2.61. Añadir o encontrar hosts.

Una vez aceptado, se mostrará el estado del host, número de *cores*, información acerca del servicio MDCE, MJS y los *workers*, en el caso que ya existan, cuando aún no han sido creados, el espacio correspondiente a dicha información estará vacía. Figura 2.62.

La siguiente opción corresponde al servicio MDCE start o stop y finalmente el Test Connectivity que es utilizado cuando existe alguna falla en el reconocimiento del dispositivo o pérdida de conectividad.

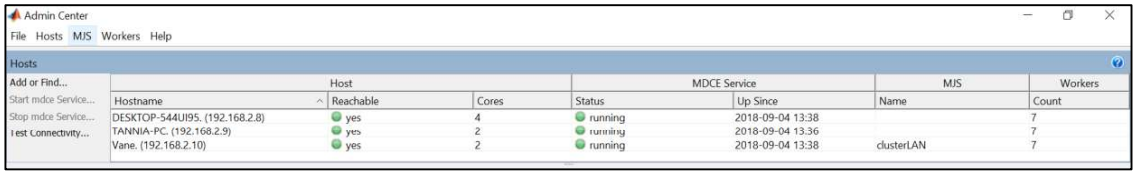


Figura 2.62. Administración de Hosts.

La sección 2, se refiere al JobManager, para eso es necesario, elegir *start,* lo cual genera una ventana según se muestra en la Figura 2.63, en la cual se ingresa el nombre asignado al clúster y se seleccion en que Host va iniciar el MJS, es posible establecer un nivel de seguridad si se desea.

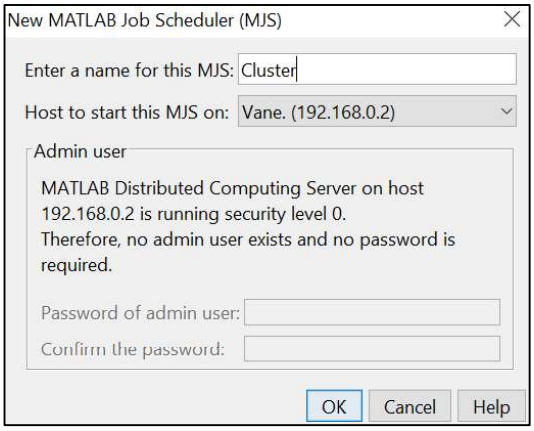


Figura 2.63. Administración de MATLAB Job Scheduler.

Finalmente, Para crear un *worker*, se va a la sección 3 de *admincenter* y a continuación se elige la opción *start*. Al presionar dicho botón aparecerá una Ventana (figura 2.64), donde se podrá elegir el número de *workers* para añadir, seleccionando en que *host* (se visualiza la lista de host que fueron descubiertos en la sección I) se van a implementar.

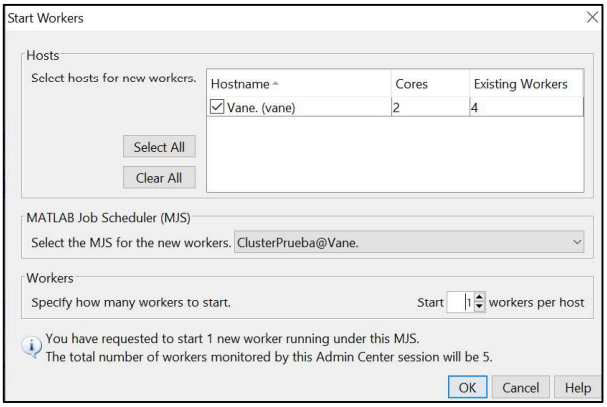


Figura 2.64. Administración de Clúster.

Después de configurar las 3 secciones del centro de administración, con las 3 PC detalladas en la creación de redes, un MJS y 21 *workers* totales, 7 por host, quedará como muestra en la figura 2.65.

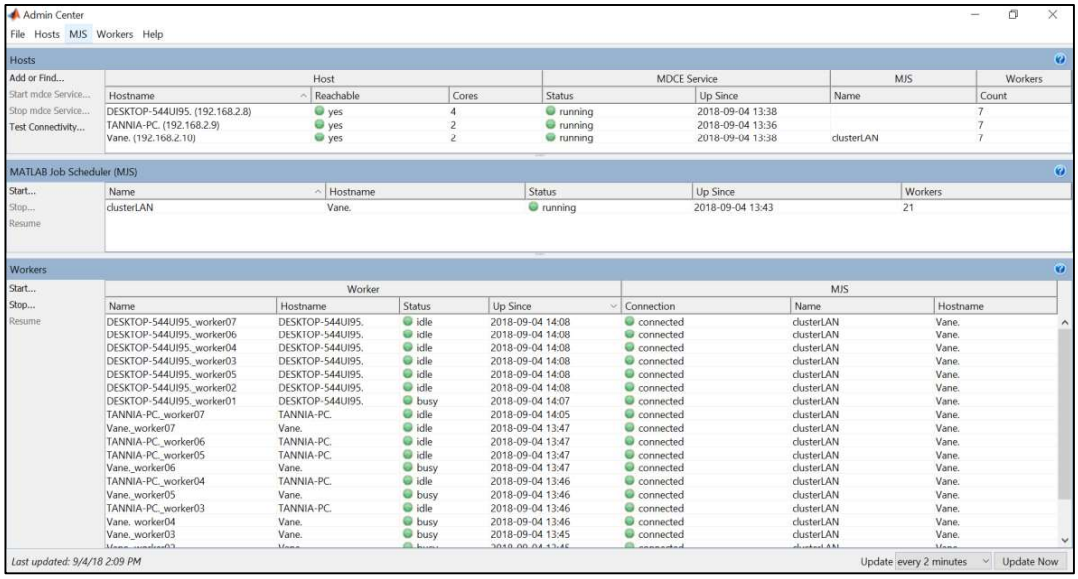


Figura 2.65. Configuración Final Clúster1.

El proceso se repite con la red Ad-hoc, ahora se muestra cómo utilizar el clúster creado desde el ambiente de trabajo de MATLAB.

La utilización del clúster se realiza de la misma manera que en el caso de clúster local, desde el Administrador de perfiles de clúster, donde se descubrirá, validará y utilizará el deseador, seleccionándolo por el nombre con el que fue creado (figura 2.66 y 2.67).

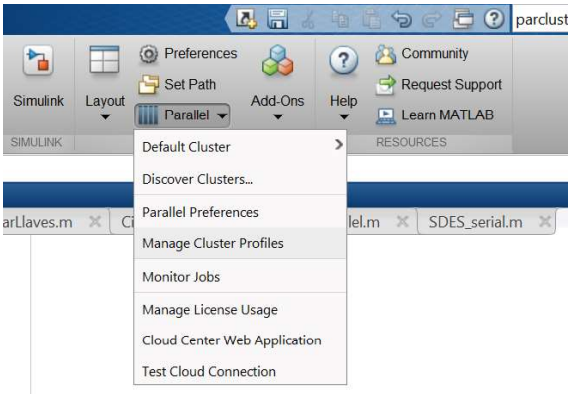


Figura 2.66. Administrador de perfiles de clústeres.

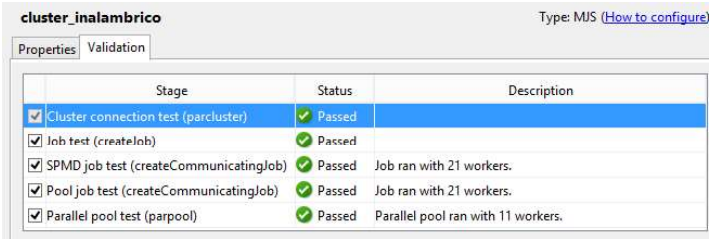


Figura 2.67. Validación de clúster de 21 trabajadores.

Creado el clúster se puede ejecutar el algoritmo paralelizado haciendo uso de todos los recursos computaciones del conjunto de PCs

3. RESULTADOS Y DISCUSIÓN

Los resultados han sido organizados de la siguiente manera:

- Interfaz gráfica
- Cifrado y Descifrado de un texto claro
- Ataque serial
- Ataque paralelo con un solo PC
- Ataque paralelo con varios PCs
- Speed-up

La interfaz gráfica se desarrolló utilizando la herramienta de MATLAB App Designer, en el ANEXO D se puede observar el diagrama de secuencia de los botones.

Para la construcción del Clúster (Alámbrico e inalámbrico) se utilizaron 3 ordenadores y 1 dispositivo de conmutación. El cliente será el computador TANNIA-PC es decir que únicamente en ese equipo se ejecutará la aplicación, para el escenario serial (figura 3.1), paralelo local (figura 3.2) y en clúster (figura 3.3).

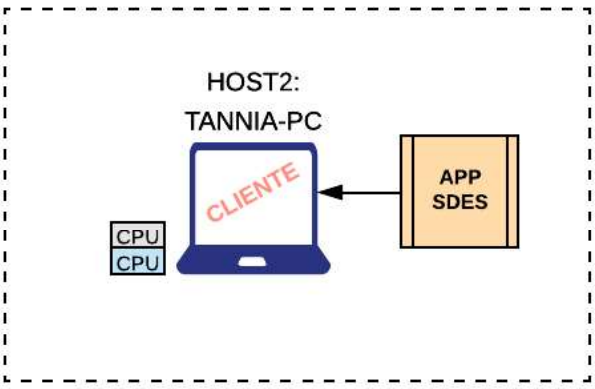


Figura 3.1. PC cliente para ejecución serial de APP

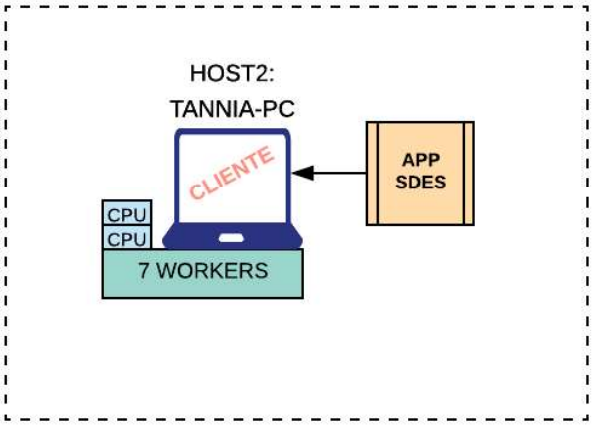


Figura 3.2. PC cliente para ejecución paralela de APP

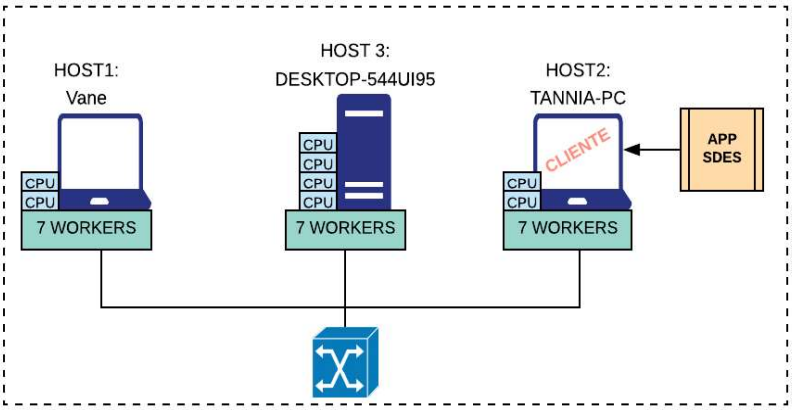


Figura 3.3. Clúster para ejecución paralela de APP

El ambiete de pruebas real se visualiza en la figura 3.4.

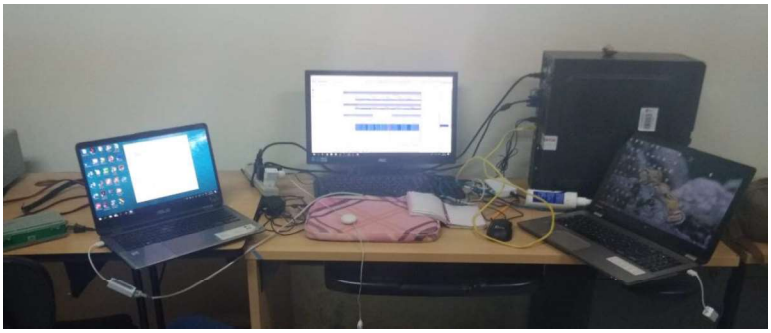


Figura 3.4. Clúster real

3.1. INTERFAZ GRÁFICA

La aplicación engloba los algoritmos de cifrado, descifrado y ataque por fuerza bruta al S-DES seriales y paralelos. Su estructura consiste en una pantalla principal que permite la elección de los dispositivos que dispone para ejecutarla (Figura 3.5). Originalmente el algoritmo de ataque por fuerza bruta se ejecuta en un clúster conformado por 3 computadores; sin embargo, debido a que los equipos cuentan con varios procesadores, se puede utilizar el paralelismo dentro de una sola computadora.

Figura 3.5. Pantalla principal aplicación S-DES.

Cualquiera que fuere la opción elegida, al ingresar, se visualizará una pantalla emergente para que el usuario especifique el número de *workers* que tendrán trabajo en la ejecución, por defecto se encuentra el número 7 (Figura 3.6).

Si el usuario ingresa un valor superior a 7 en la opción “1 computador” o superior a 21 en “Clúster” se presenta un error que dará a conocer que los perfiles de clústeres están especificados para esos valores como máximo.

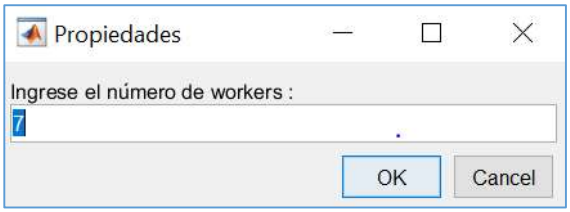


Figura 3.6. Propiedades de Clúster

Al confirmar el número de *workers*, se obtiene un mensaje en la pantalla principal que mantiene un color amarillo hasta que se haya iniciado el *parallel pool* con el número indicado, esto puede tomar varios segundos incluso pocos minutos, dependiendo de los programas que se encuentren ejecutando en el o los equipo, por lo que se recomiendo que se cierre cualquier cosa que pueda ralentizar la ejecución (Figura 3.7).

Figura 3.7. Inicio del Parallel Pool.

Posteriormente se muestra una pantalla que permite navegar entre las acciones de cifrar, descifrar, atacar, crear un archivo nuevo para procesarlo, exportar un archivo en formato txt o Word pudiendo también imprimirlo. Como punto común entre las paginas CIFRAR, DESCIFRAR Y ATAQUE (Figura 3.8), se tiene la sección de búsqueda, donde el usuario podrá localizar el archivo que desea, mostrando el path y el contenido, además los métodos siempre serán serial o paralelo. El botón estado comunica al usuario el proceso de la acción seleccionada, de manera que cuando esté en rojo, aún no ha iniciado ningún proceso, en naranja el proceso está pendiente y finalmente en verde, la solución estará a la vista.

Si se necesita hacer un cambio en el tamaño del clúster seleccionado, puede regresar atrás mediante el ícono  .

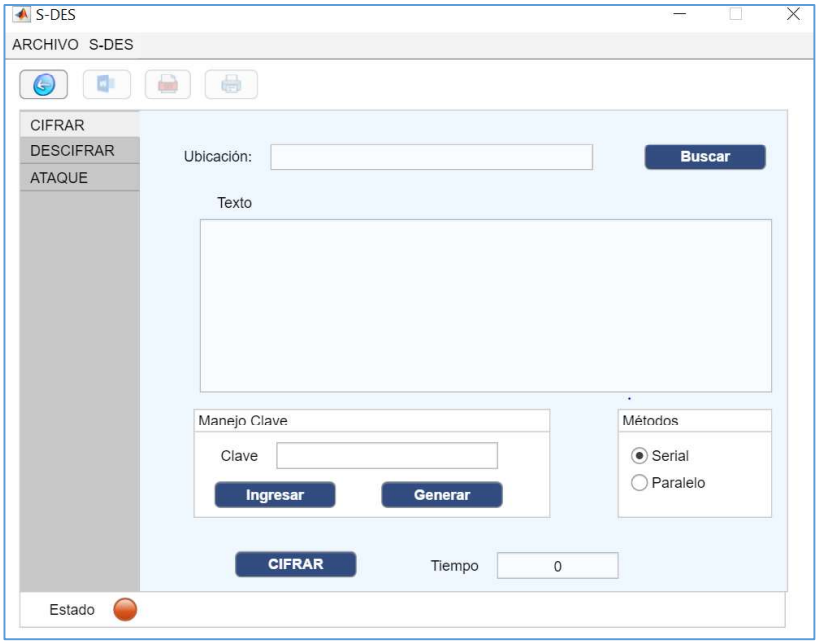


Figura 3.8. Pantalla principal de la aplicación.

A continuación, se describirán las pruebas de funcionamiento en cada entorno seleccionado y sus respectivos tiempos de ejecución, serial y con variaciones dentro del ambiente paralelo para cada caso. Se consideran 2 casos principales para evaluar: texto corto que corresponde a un conjunto de 10 líneas y texto largo: 3 páginas aproximadamente.

3.2. CIFRADO Y DESCIFRADO DE UN TEXTO CLARO

Se ha ejecutado el cifrado y descifrado serial y paralelo en los 2 textos mencionados anteriormente, solo se consideran los casos serial, paralelo con 7 *workers* y paralelo en clúster con 21 *workers*.

En la Figura 3.9, 3.10 y 3.11 se puede observar la interfaz gráfica que muestra las 3 opciones que contiene la aplicación, primero se busca un archivo, posterior a eso se ingresa una clave de 10 bits en binario en el caso de usar el cifrado, esta puede ser generada aleatoriamente si se desea, luego se elige el método, pudiendo ser serial o paralelo (en un PC o varios), obteniendo el texto cifrado/descifrado en el recuadro superior, el mismo que puede ser exportado.

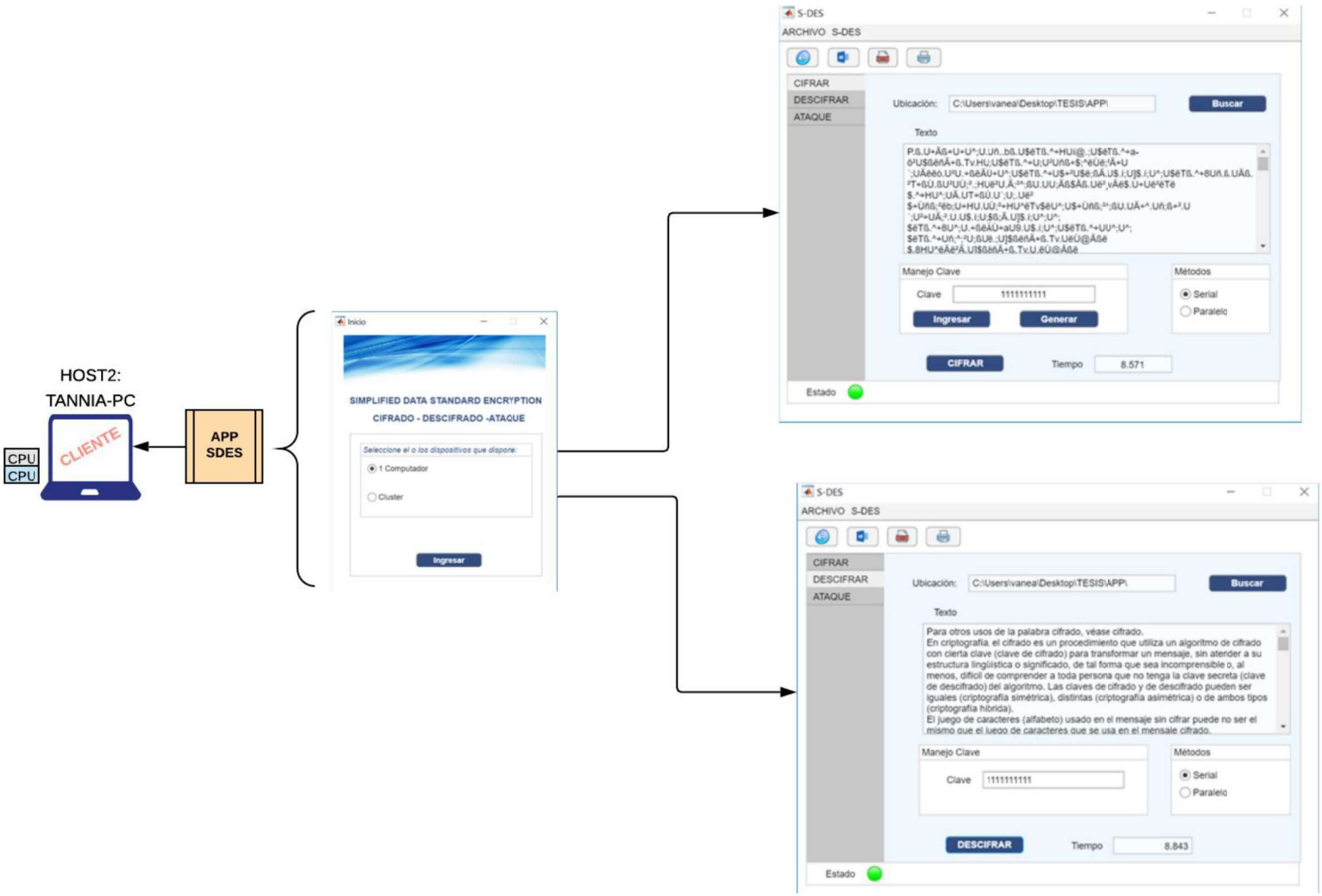


Figura 3.9. Cifrado/Descifrado serial de un archivo de texto

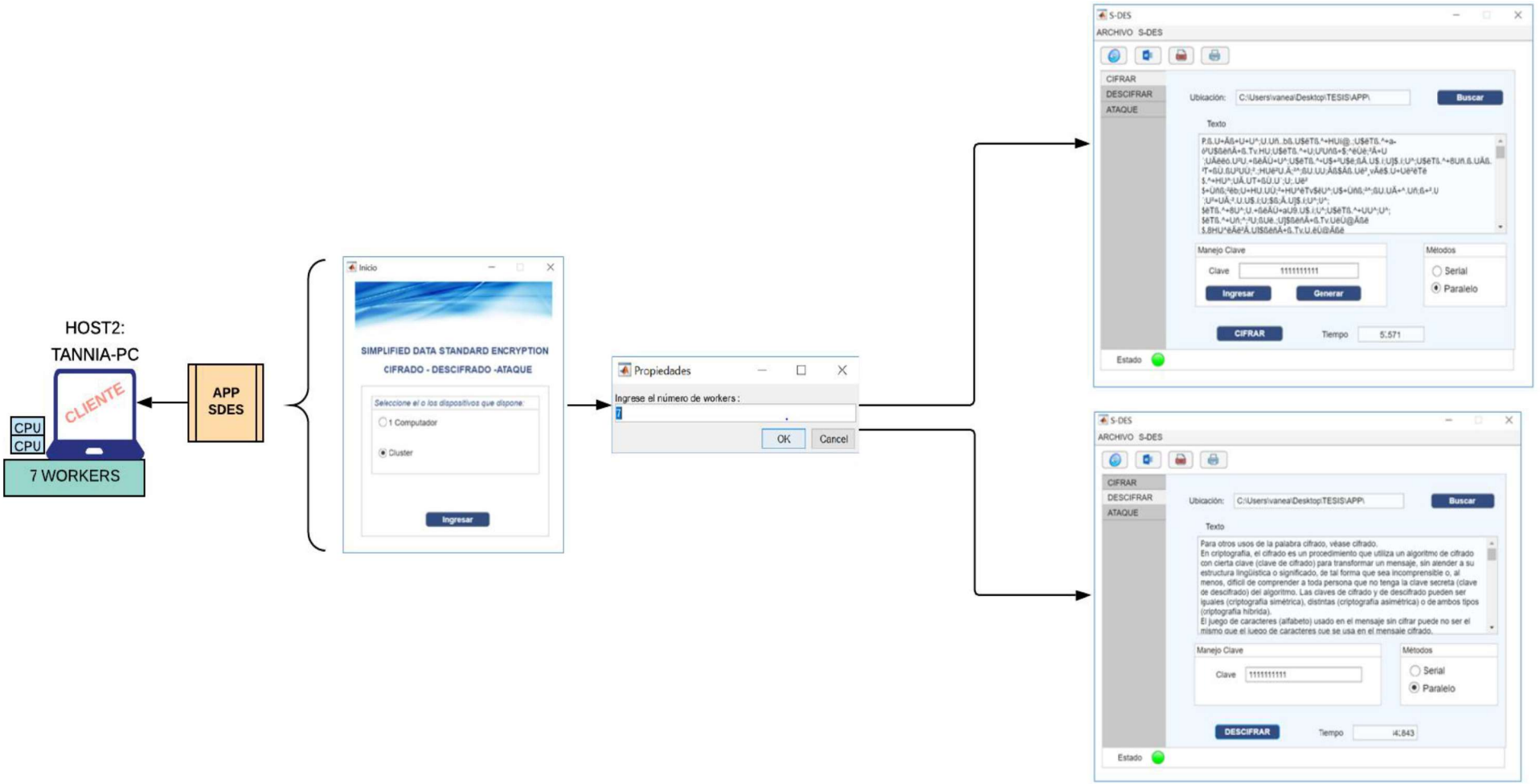


Figura 3.10. Cifrado/Descifrado paralelo en un PC de un archivo de un archivo de texto

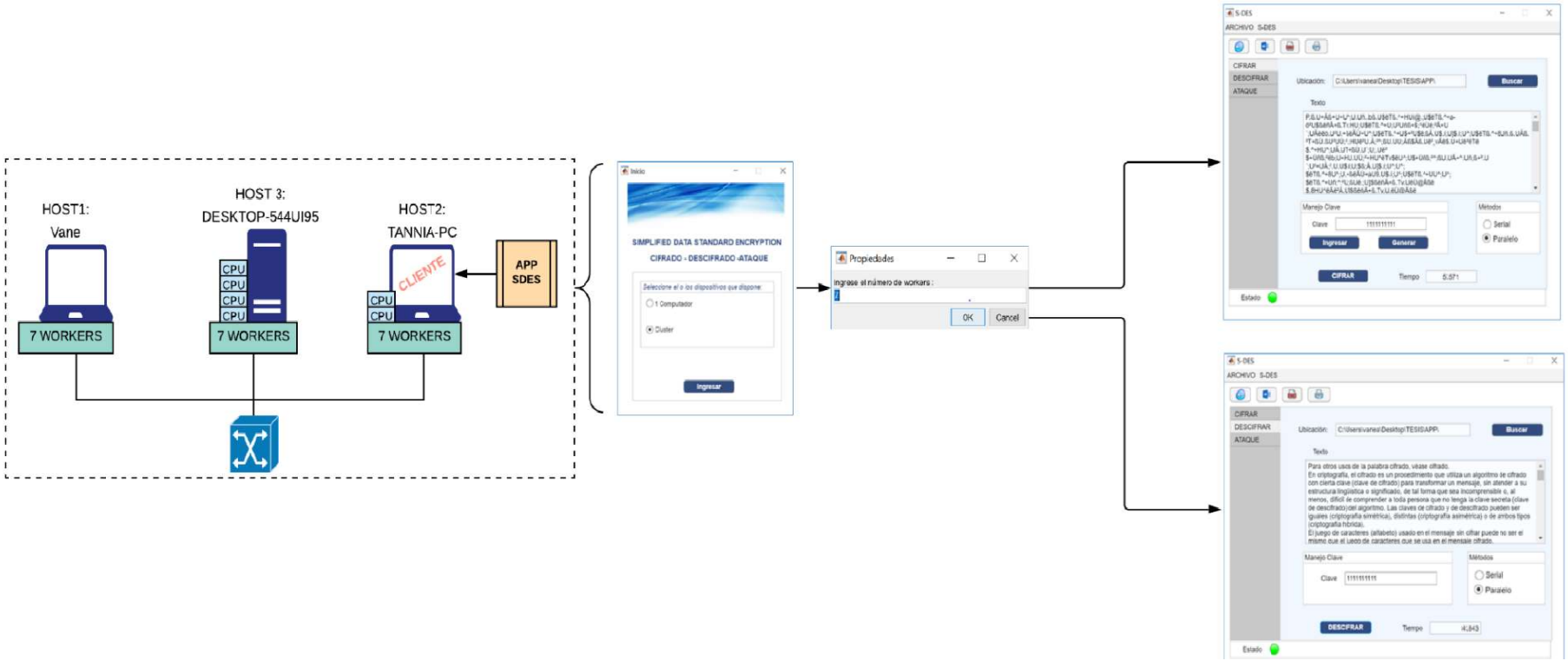


Figura 3.11. Cifrado/Descifrado paralelo en un clúster de un archivo de un archivo de texto.

En la Tabla 3.1 se puede observar los tiempos que se tarda en cifrar un archivo secuencialmente, respecto al texto corto, no sobrepasan los 4.42 segundos, mientras que, para texto largo, se alcanza casi a los 7 segundos.

Tabla 3.1. Tiempos de ejecución cifrado S-DES serial.

Procesamiento Serial			
Texto Corto		**Texto Largo**	
Ejecución 1	**Ejecución 2**	**Ejecución 1**	**Ejecución 2**
4.42 s	3.57 s	6.92 s	6.90 s

En la Tabla 3.2 se aprecian los tiempos paralelos para los mismos casos, texto corto y largo, se reduce en un porcentaje superior al 50 % para texto corto; sin embargo, en el texto largo otra es la situación, el valor de reducción se asemeja al 34 %, se observa que, debido a la duración demasiado corta del proceso, tal vez no se podría apreciar demasiado la mejora que existe en la ejecución.

Tabla 3.2. Tiempos de ejecución cifrado S-DES paralelo LOCAL

Procesamiento Paralelo Local			
Texto Corto		**Texto Largo**	
Ejecución 1	**Ejecución 2**	**Ejecución 1**	**Ejecución 2**
1.94 s	1.90 s	4.50 s	4.75 s

Ahora se presentan los tiempos obtenidos al ejecutar el cifrado dentro del clúster con 21 en la Tabla 3.3. Se puede ver que para el caso de texto corto solo existe una mejora del 22% aproximadamente. En la ejecución del cifrado para texto largo, la mejora es del 66%, esto respecto a los tiempos de la Tabla 3.1. Ahora se compara con los tiempos obtenidos en la ejecución secuencial y se puede ver que los valores de mejora se acercan al 66 % en texto corto y 78% en texto largo.

Tabla 3.3. Tiempos de ejecución cifrado S-DES paralelo CLÚSTER

Procesamiento Paralelo Clúster			
Texto Corto		**Texto Largo**	
Ejecución 1	**Ejecución 2**	**Ejecución 1**	**Ejecución 2**
1.50 s	1.52 s	1.88 s	1.45 s

Los mismo sucede para el descifrado de un archivo de texto, en la interfaz gráfica se puede buscar el archivo deseado y obligatoriamente se necesita conocer la clave con la que este fue cifrado ya que, si no, no se obtendrá el texto original.

Se consideran los mismos escenarios especificados para el cifrado, la Tabla 3.4 muestra los tiempos de ejecución con el algoritmo secuencial.

Tabla 3.4. Tiempos de ejecución descifrado S-DES serial.

Procesamiento Serial			
Texto Corto		**Texto Largo**	
Ejecución 1	**Ejecución 2**	**Ejecución 1**	**Ejecución 2**
3.73 s	3.69 s	7.74 s	7.28 s

En la Tabla 3.5 se puede apreciar como los tiempos disminuyen al utilizar todos los procesadores que tienen un equipo paralelamente.

Tabla 3.5. Tiempos de ejecución descifrado S-DES paralelo LOCAL

Procesamiento Paralelo Local			
Texto Corto		**Texto Largo**	
Ejecución 1	**Ejecución 2**	**Ejecución 1**	**Ejecución 2**
2.13 s	2.27 s	3.87 s	3.73 s

El contraste final se puede ver en la Tabla 3.6 donde se puede ver los tiempos obtenidos al ejecutar el algoritmo de descifrado paralelo en 3 equipos.

Tabla 3.6. Tiempos de ejecución cifrado S-DES paralelo CLÚSTER

Procesamiento Paralelo Clúster			
Texto Corto		**Texto Largo**	
Ejecución 1	**Ejecución 2**	**Ejecución 1**	**Texto Largo**
0.83 s	0.91 s	1.4 s	1.33 s

3.3. ATAQUE SERIAL O SECUENCIAL

Las pruebas del algoritmo consisten en varias ejecuciones tomando un texto cifrado, sin conocer la clave, se ha considerado los textos; de tamaño corto y largo, cifrados previamente. Se toman los tiempos cuando solo actúa un procesador (secuencialmente), esto se va incrementando, cuando existen 2, 3, 4, 5 ,6 y 7 *workers* en un solo equipo. Dentro del clústeres se puede incrementar de 2 a 21 *workers*.

La Figura 3.12. muestra el resultado satisfactorio de un ataque, donde aparece el texto original, la clave con la que fue cifrado y el tiempo que le tomó lograrlo, para el caso serial.

Se obtendrán los resultados de la Tabla 3.7, donde se visualiza una diferencia de apenas 1 segundo como máximo entre los tiempos tomados en los 2 tamaños de textos, si bien el texto largo sobrepasa el tamaño del texto corto casi por 6 veces, la diferencia del ataque entre estos tomando el promedio de las 2 ejecuciones, es de casi 4 segundos. Esto sucede por lo explicado en la elaboración del algoritmo: solo se evalúan 80 caracteres del texto cifrado, en el caso de que se evaluaran todos los caracteres que componen el texto en cuestión, el tiempo de ataque aumentaría notoriamente, en el caso de texto largo se ha realizado una prueba y se obtiene la clave con la que se ha cifrado en aproximadamente 1 hora.

Tabla 3.7. Tiempos de ejecución del algoritmo de ataque al S-DES serial.

Procesamiento Serial			
Texto Corto		**Texto Largo**	
Ejecución 1	**Ejecución 2**	**Ejecución 1**	**Ejecución 2**
87.34 s	86.45 s	90.59 s	91.04 s

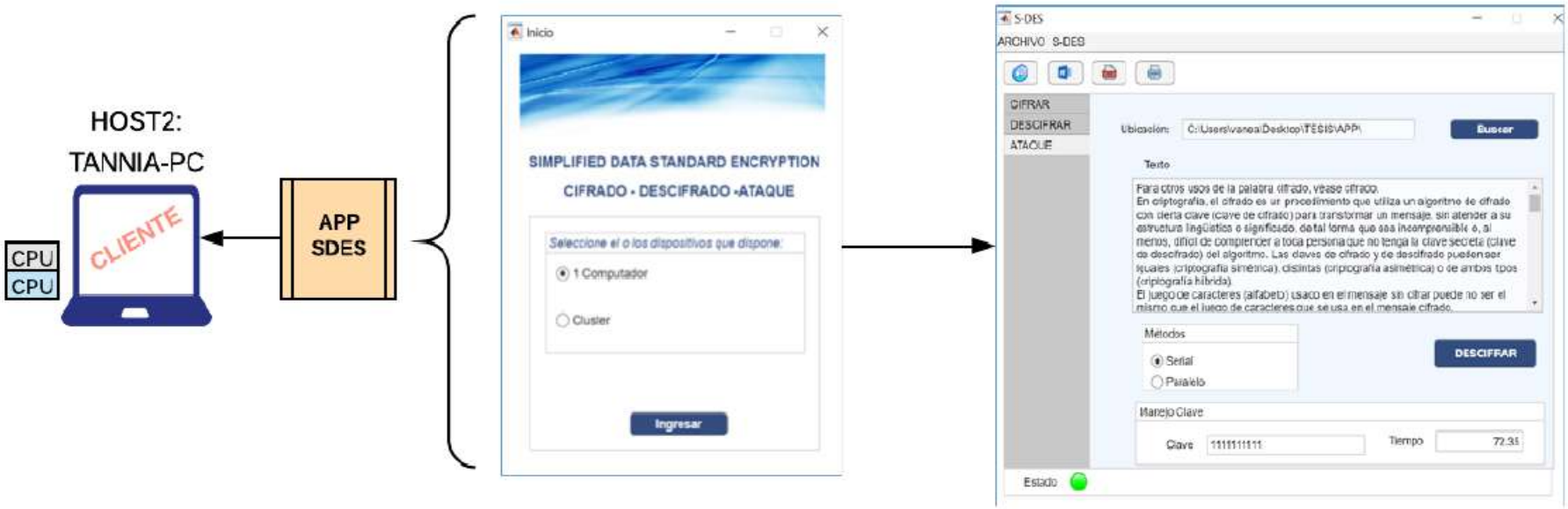


Figura 3.12. Ataque serial de un archivo de texto.

3.4. ATAQUE PARALELO CON UN SOLO PC

En el procesamiento paralelo local se obtienen los resultados que se muestran en la Tabla 3.8 utilizando el algoritmo paralelizado con PARFOR y en la Tabla 3.9 con el algoritmo paralelizado con SPMD.

Tabla 3.8. Tiempos de ejecución del algoritmo de ataque al S-DES paralelo con PARFOR.

Procesamiento Paralelo-PARFOR							
Texto Corto				**Texto Largo**			
Ejecución 1		**Ejecución 2**		**Ejecución 1**		**Ejecución 2**	
Workers	**Tiempo(s)**	***Workers***	**Tiempo(s)**	***Workers***	**Tiempo(s)**	***Workers***	**Tiempo(s)**
2	48.61	2	49.27	2	53.27	2	51.32
3	45.5	3	45.42	3	48.42	3	50.6
4	44.56	4	44.11	4	47.11	4	46.03
5	43.9	5	43.74	5	49.74	5	48.38
6	43.1	6	42.7	6	49.17	6	47.96
7	42.68	7	42.17	7	45.7	7	45.8

Si se compara respecto al tiempo que le tomó el ataque de manera serial, se puede apreciar una notable reducción en tiempo, cuando se utiliza la mínima posibilidad de trabajadores se considera una mejora de aproximadamente 44% en texto corto y 42% en texto largo, mientras que con 7 trabajadores la mejora es de 51% y 50% respectivamente.

Al aumentar los trabajadores que ejecutan el algoritmo se nota una cierta mejora, pero no demasiado, esto depende de cómo se ha distribuido el trabajo entre los procesadores del equipo en el que se realiza la ejecución, no se tiene el control de dicha distribución, así que no se pueden predecir tiempos, así como la tabla indica estos valores, en una siguiente ejecución respetando los parámetros para estas pruebas se puede obtener tiempos diferente.

Ahora para tener algo de control respecto a la distribución del trabajo, se presentan los tiempos obtenidos con SPMD, que como se explicó anteriormente permite condicionar que un laboratorio (worker) específico haga una tarea determinada, así que puede que un trabajador haya encontrado la solución a los pocos segundos, sin embargo, hasta que se comunique con los otros trabajadores para indicarles esto, el tiempo aumenta, por esto la

diferencia de tiempos con respecto a los de la Tabla 3.9. Por este motivo para la aplicación se eligió el algoritmo que contiene PARFOR.

Tabla 3.9. Tiempos de ejecución del algoritmo de ataque al S-DES paralelo con SPMD.

Procesamiento Paralelo-SPMD							
Texto Corto				**Texto Largo**			
Ejecución 1		**Ejecución 2**		**Ejecución 1**		**Ejecución 2**	
Workers	**Tiempo(s)**	***Workers***	**Tiempo(s)**	***Workers***	**Tiempo(s)**	***Workers***	**Tiempo(s)**
2	54.8	2	61.07	2	57	2	63.61
3	52.12	3	59.43	3	51.9	3	59.34
4	52.6	4	57.05	4	58.88	4	59.04
5	50.37	5	57.9	5	50.75	5	58.5
6	58.65	6	53.5	6	50.4	6	57.43
7	58.12	7	52.53	7	59.71	7	55.35

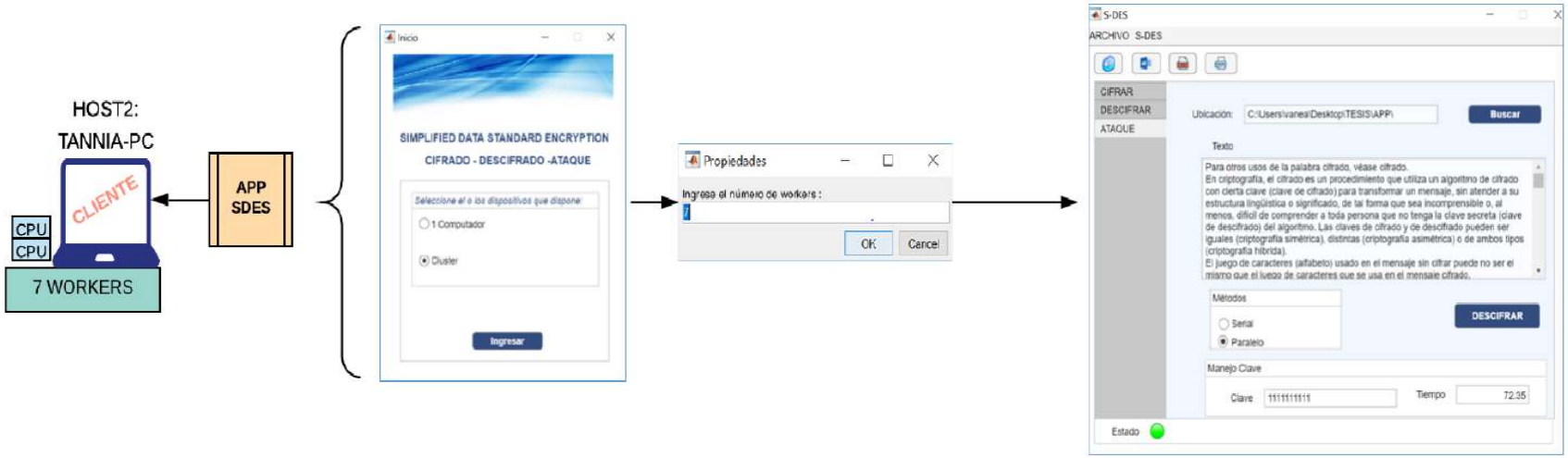


Figura 3.13. Ataque paralelo en un PC de un archivo de texto

3.5. ATAQUE PARALELO CON VARIOS PCS

Para el procesamiento paralelo se designan 2 escenarios de prueba:

- Red cableada

- Red Ad-hoc.

3.5.1 CLÚSTER LAN

Los tiempos obtenidos en la ejecución de algoritmo paralelo con PARFOR se denotan en la Tabla 3.10. Si bien el clúster estaría ejecutándose desde el trabajador 15 hasta el 25, los tiempos anteriores se toman con el objetivo de ver el funcionamiento del clúster.

Se ha dividido en secciones, la primera es correspondiente a un solo equipo, pero esto depende del Job Manager, en este caso el primer equipo de la lista de host es DESKTOP-544UI95. Los tiempos se reducen conforme se aumentan los trabajadores, se realizan 3 ejecuciones ya que la primera ejecución tiene un valor diferente a las 2 siguientes, esto porque el par pool ha sido recién iniciado.

En la segunda sección, el algoritmo se ejecuta en un clúster de 2 equipos, siguiendo el orden en que se encuentren los hosts de la lista, los tiempos disminuyen, pero cuando trabajan 11 y 12 *workers* se dan valores altos, respecto al anterior, esto sucede ya que el Job Manager decidió combinar los 2 últimos equipos de la lista: VANE y TANNIA-PC que como se indicó tienen características más bajas respecto a procesador y RAM que DESKTOP-544UI95. Cuando 13 *workers* trabajan los tiempos se estabilizan, disminuyendo, hasta llegar a la utilización de los 21 *workers* con los 3 equipos, el tiempo promedio para texto corto 11.17 segundos y para texto largo 11.41 segundos, esto es una mejora notable respecto al tiempo de 42 y 45 segundos utilizando el clúster local y aún más notorio si se compara con el tiempo de ejecución serial de 86 y 90 segundos.

La mejora corresponde a un 87% en texto corto y texto largo.

Tabla 3.10. Tiempos de ejecución del algoritmo de ataque al S-DES en clúster LAN.

Clúster LAN						
Procesamiento Paralelo (PARFOR)						
Workers	Texto Corto			Texto Largo		
	Ejecución 1	Ejecución 2	Ejecución 3	Ejecución 1	Ejecución 2	Ejecución 3
2	28.22	27.59	27.29	30.34	26.52	26.54
3	22.1	22.76	22.56	22.66	22.9	22.87
4	21.89	21.7	21.99	22.02	21.98	22.07
5	20.11	20.62	20.35	21.5	21.78	21.2
6	20	19.45	19.71	21.04	20.86	20.69
7	19.4	19.02	19	20.71	20.4	20.1
8	18.56	18.41	18.29	19.54	19.65	19.84
9	18.32	18.2	18.06	19	19.02	19.45
10	17.24	17.39	17.48	18.7	18.38	18.43
11	20.21	20.91	20.53	21.76	21.89	21.48
12	21.66	21.53	22	24.02	23.27	23.76
13	16.87	16.52	16.8	17.93	18.04	18.22
14	16.78	16.9	16.07	17.01	16.98	17.03
15	15.50	15.20	15.31	16.07	16.25	16.1
16	13.96	14.04	14.13	14.38	14.29	14.4
17	13.67	13.82	13.49	13.92	13.7	13.68
18	13.02	13.31	13.29	13.5	13.16	13.26
19	12.24	12.27	12.32	12.87	12.57	12.42
20	12.07	11.82	11.56	12.38	11.96	11.77
21	11.26	11.16	11.10	11.46	11.58	11.20

Figura 3.14. Ataque paralelo en un cúster de un archivo de tex

3.5.2. CLÚSTER AD-HOC

Los tiempos obtenidos en la ejecución de algoritmo paralelo con PARFOR se denotan en la Tabla 3.11.

Debido a la conexión inalámbrica, los tiempos aumentan; sin embargo, la mejora sigue existiendo. El porcentaje de reducción de tiempo respecto al texto corto y largo es aproximadamente de 85%.

Tabla 3.11. Tiempos de ejecución del algoritmo de ataque al S-DES en clúster AD-HOC

Cluster Inalámbrico						
Procesamiento Paralelo(PARFOR)						
Workers	**Texto Corto**			**Texto Largo**		
	Ejecución 1	**Ejecución 2**	**Ejecución 3**	**Ejecución 1**	**Ejecución 2**	**Ejecución 3**
2	29.02	28.4	28.12	30.89	28.45	27.99
3	23.31	22.49	22.51	23.79	22.81	22.76
4	22	21.98	21.46	22.67	21.98	22.07
5	21.74	20.73	20.35	22.02	21.88	21.72
6	20.39	20.13	19.98	20.98	20.68	20.78
7	20.01	19.78	19.63	20.81	20.42	20.31
8	19.11	18.69	18.88	19.65	19.55	19.54
9	18.92	18.51	18.58	19.3	19.02	18.9
10	18.04	17.69	17.59	18.65	18.78	18.49
11	17.95	17.13	17.11	18.27	18.07	17.96
12	17.47	17	17.09	17.99	17.8	17.76
13	16.95	16.42	16.78	17.83	18	17.76
14	16.68	16.29	16.27	17.12	16.95	16.91
15	15.30	15.00	15.50	16.07	16.25	16.1
16	14.97	14.75	14.02	15.33	15.09	14.9
17	13.36	13.59	13.05	13.92	13.71	13.67
18	13.22	13.13	13.19	13.56	13.34	13.36
19	12.47	13.46	12.52	13.02	13.11	12.93
20	12.15	11.41	12.10	12.65	12.70	12.80
21	12.02	11,56	11.56	12.14	11.79	11.56

3.6. RESULTADOS DE TIEMPOS CON CLAVES DIFERENTES

Como se especificó, los tiempos obtenidos en el ataque fueron el resultado de un texto cifrado con la clave de mayor dificultad, la que contiene todos sus bits “1s”, pero ¿qué pasa si la clave es más “sencilla” ?, en la Tabla 3.12 se indica como varía los tiempos.

Se ha tomado solamente el texto largo, y el algoritmo paralelo con PARFOR para esta prueba, además que se utiliza todos los trabajadores, tanto para el clúster local (7), como para el clúster LAN (21).

Tabla 3.12. Tiempos con claves diferentes serial y paralelo.

Valor Binario	Serial (s)	Paralelo local (s)	Paralelo Clúster (s)
"0000000001"	8.24	45.93	11.23
"0001100100"	15.89	43.82	12.01
"0011001000"	15.57	46.73	11.43
"0100101100"	30.72	47.5	11.97
"0110010000"	39.09	45.41	11.6
"0111110100"	46.02	46.16	12.6
"1001011000"	54.98	47.28	13.01
"1010111100"	60.92	47.05	12.8
"1100100000"	69.1	41.96	11.76
"1110000100"	77.59	42.04	11.34
"1111111111"	87	45.84	11.45

Si se compara los tiempos seriales con los paralelos en la primera clave se puede decir que paralelizar un algoritmo no es eficiente, más aun cuando se verifica los tiempos obtenidos con el clúster local, en la segunda clave ya se puede observar una ventaja en el tiempo obtenido con el clúster; sin embargo, localmente el tiempo sigue siendo malo, conforme la clave se dificulta para "1001011000" se observa que los 2 clústeres paralelos ya tienen una beneficio notorio y así continua hasta la clave con más dificultad.

Si se analiza la columna de Paralelo local, se puede apreciar que los tiempos incrementan partiendo de 41 segundos, pero no sobrepasan los 47, algo parecido sucede en la columna Paralelo Clúster, los tiempos oscilan entre 11 y 12. En la columna serial por supuestos los tiempos aumentan con cada clave. Conforme la longitud de la clave aumente, el número de claves también lo hará, un algoritmo paralelo será más eficiente cuando más grande sea la clave.

3.7. SPEED-UP

Como se explicó en la parte teórica la speed-up es una de las medidas principales para evaluar un algoritmo paralelo, a continuación, se presentan los datos tomados para el

cálculo de la Speed-up absoluta y relativa, se utilizan valores de ts (tiempo de ejecución obtenido por el mejor algoritmo serial) 66,44 s para texto largo y 63.65 para texto corto.

Estas pruebas han sido realizadas para los siguientes casos:

-Paralelo local (un PC) speed-up absoluta y relativa

-Paralelo en un clúster LAN speed-up absoluta y relativa

-Paralelo en un clúster inalámbrico speed-up absoluta y relativa

3.7.1. SPEED-UP ABSOLUTA Y RELATIVA EN UN PC

Para el cálculo del Speed-up absoluto y relativo se ha tomado en cuenta 2 algoritmos: el secuencial y paralelo, pero para el cálculo de speed-up absoluto se evaluó respecto al mejor algoritmo secuencial.

Estos algoritmos se diferencian debido a que el mejor secuencial no genera una matriz de claves posibles, sino que evalúa una a una sin necesidad de almacenarlas; sin embargo, este no se podría paralelizar así, ya que los datos se verían obligados a estar en cola, a diferencia del secuencial normal que utiliza la matriz de claves de modo que, al ser paralelizado, dicha matriz se distribuye en cada trabajador y estos valores serán evaluados separados, pero simultáneamente.

En la tabla 3.13 se muestran los valores resultantes para la speed-up absoluta y relativa de un texto largo, con un número incremental de *workers* de 2 a 7, donde n representa los trabajadores utilizados en el clúster.

Se observa la variación de speed-up absoluta para el valor inicial de 2 trabajadores es 1.27 y SpeddUp relativa 1.68 a partir de este se puede ver como se incrementa conforme el número de trabajadores; sin embargo, cuando para n=5 decae, vuelve a subir en 6 y finalmente en 7 vuelve a decaer.

Tabla 3.13. Speed-up absoluta y relativa local en texto largo.

Texto Largo		
n	**Speed-up Absoluta**	**Speed-up Relativa**
2	1.27	1.68
3	1.34	1.77
4	1.44	1.91
5	1.35	1.79
6	1.40	1.85
7	1.34	1.77

La figura 3.15 muestra la gráfica respecto a los valores descritos de speed-up Absoluta y relativa para un texto largo, el incremento o decremento se mantiene en las 2 líneas(se observan 2 picos para n=4 y n=6) pero para speed-up absoluta los valores son mayores; sin embargo, no siempre se va poder utilizar el mejor algoritmo serial, por lo cual los valores reales serían de speed-up relativa.

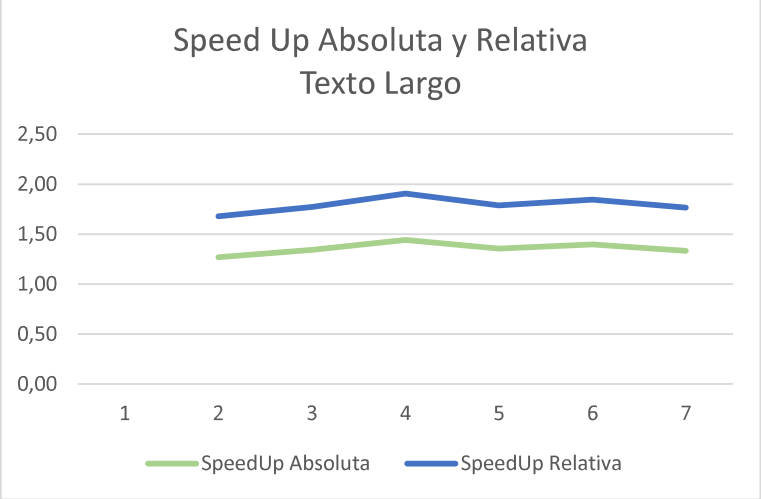


Figura 3.15. Speed-up absoluta y relativa local en texto largo.

El mismo proceso se ha realizado para un texto corto, la tabla 3.14 se observa la variación de speed-up absoluta para el valor inicial de 2 trabajadores es 1.28 y speed-up relativa 1.61 a partir de este se puede ver como se incrementa hasta n=3, en n=4 decae, pero continua con valores mayores hasta llegar a n=7.

Tabla 3.14. Speed-up absoluta local en texto corto.

Texto Corto		
n	**Speed-up Absoluta**	**Speed-up Relativa**
2	1.28	1.61
3	1.50	1.89
4	1.46	1.84
5	1.56	1.96
6	1.60	2.01
7	1.63	2.05

En la Figura 3.16 la gráfica de la speed-up absoluta y relativa para un texto corto tiene un comportamiento similar a un texto largo, pero no es igual ya que los picos existentes no ocurren cuando se trabaja con el mismo número de *workers*, además los valores aumentan por la diferencia de longitud del texto procesado, las 2 curvas mantienen la tendencia.

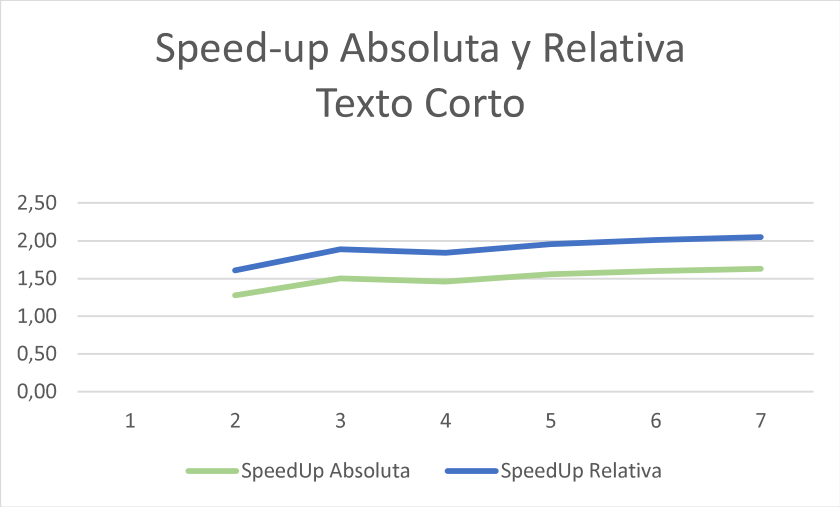


Figura 3.16. Speed-up absoluta y relativa local en texto corto.

3.7.2. SPEED-UP ABSOLUTA Y RELATIVA EN UN CLUSTER LAN

La tabla 3.15 recopila los datos necesarios para el cálculo de la speed-up absoluta y relativa cuando se ejecuta el algoritmo paralelo en un clúster LAN para un texto largo, los valores de speed-Up absoluta oscilan entre 2 y 6 mientras que de speed-up relativa entre 3 y 8.

La figura 3.17 muestra un incremento de speed-up en el intervalo de 2 a 10 *workers*, al llegar a 11 y 12 se ve una caída considerable, luego a partir de 13 hasta 21 la vuelve alcanzar un punto alto. Como ya se explicó en la toma de tiempos, el Job Manager distribuye el trabajo de una manera no especificada, con 10 y 11 *workers* realizó la distribución entre los equipos de menor prestación, mientras que el de mejor prestación estuvo sin realizar trabajo, a diferencia de cuando se trabajó con 14 hasta 21 donde actuaban todos los equipos. Ambas curvas mantienen la tendencia con valores superiores para la speed-up relativa.

Tabla 3.15. Speed-up absoluta y relativa en un clúster LAN en texto largo.

Texto Largo		
n	**Speed-up Absoluta**	**Speed-up Relativa**
2	2.39	3.16
3	2.91	3.85
4	3.02	3.99
5	3.09	4.09
6	3.18	4.21
7	3.26	4.30
8	3.38	4.46
9	3.47	4.58
10	3.59	4.75
11	3.06	4.05
12	2.81	3.71
13	3.68	4.86
14	3.91	5.16
15	4.12	5.44
16	4.63	6.12
17	4.83	6.38
18	4.99	6.60
19	5.26	6.96
20	5.52	7.30
21	5.82	7.70

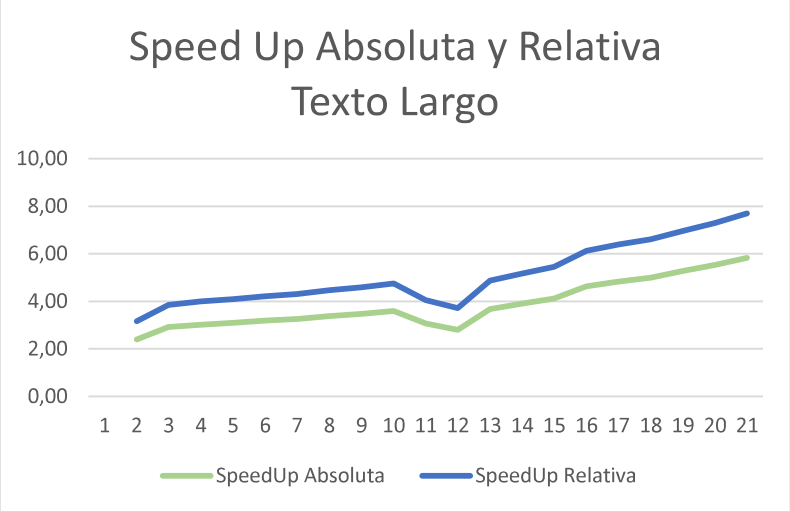


Figura 3.17. Speed-up absoluta y relativa en un clúster LAN en texto largo.

La tabla 3.16 corresponde a los valores de speed-up absoluta y relativa para texto corto en el mismo escenario de prueba, los valores son semejantes a los de la tabla 3.15, entre 2 y 6. , el resultado es similar que cuando se ejecutó en texto largo en relación a subidas y bajadas (intervalo de 11 a 12 trabajadores); sin embargo, el peor valor es de 2.30 para la absoluta y 2.89 para relativa con 2 trabajadores y el mejor de 5,50 y 7.16 respectivamente, esto dado por la diferencia de longitud de la información ya que por ser menor se obtienen tiempos más bajos.

Tabla 3.16. Speed-up absoluta y relativa en un clúster LAN en texto corto.

Texto Corto		
n	**Speed-up Absoluta**	**Speed-up Relativa**
2	2.30	2.89
3	2.83	3.56
4	2.91	3.66
5	3.13	3.93
6	3.23	4.06
7	3.33	4.18
8	3.46	4.34
9	3.50	4.40
10	3.66	4.61
11	3.10	3.89
12	2.93	3.68
13	3.80	4.78
14	3.84	4.82
15	4.15	5.22
16	4.53	5.70
17	4.66	5.86
18	4.82	6.06
19	5.18	6.52
20	5.39	6.77
21	5.70	7.16

En la Figura 3.18 se puede observar los valores (Tabla 3.16) plasmados a través de una gráfica.

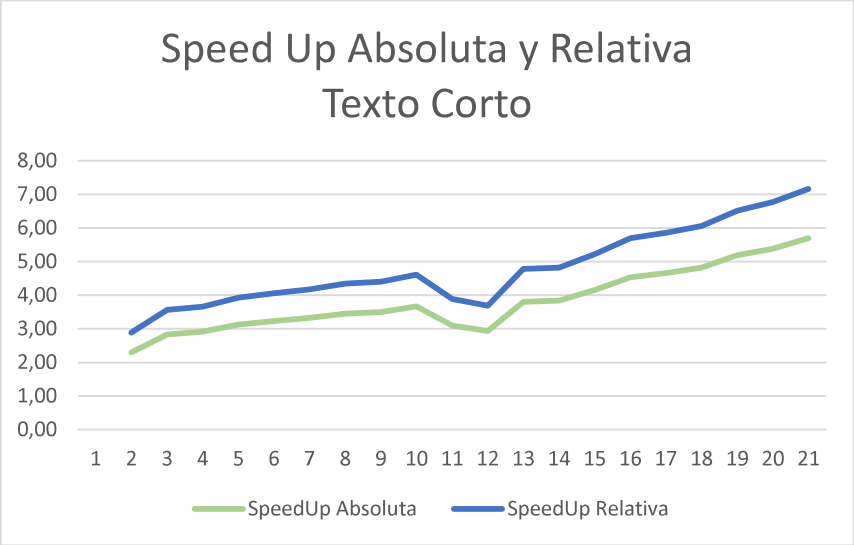


Figura 3.18. Speed-up absoluta y relativa en un clúster LAN en texto corto.

3.7.3. SPEED-UP ABSOLUTA Y RELATIVA EN UN CLÚSTER INALÁMBRICO

La tabla 3.17 se presentan los valores de Speed-up absoluta y relativa cuando se ejecuta el algoritmo paralelo en un clúster inalámbrico para un texto largo, los valores de Speed-up absoluta oscilan entre 2 y 6 mientras que de Speed-up relativa entre 3 y 8 de la misma forma que para un clúster LAN con diferencias muy pequeñas respecto a valores decimales, hasta n=10, en este caso no hay variación de decremento sino que los valores suben hasta llegar a n=21, esto está justificado por la distribución de trabajo en el clúster.

En la Figura 3.19 se puede observar los valores (Tabla 3.17) plasmados a través de una gráfica donde las 2 curvas se mantienen en incremento durante todo el trayecto hasta llegar a n=21.

Tabla 3.17. Speed-up absoluta y relativa en un clúster inalámbrico con texto largo.

Texto Largo		
n	**Speed-up Absoluta**	**Speed-up Relativa**
2	2.28	3.02
3	2.87	3.80
4	2.99	3.95
5	3.04	4.02
6	3.19	4.22
7	3.24	4.28
8	3.39	4.49
9	3.48	4.60
10	3.56	4.71
11	3.67	4.85
12	3.72	4.92
13	3.72	4.92
14	3.91	5.17
15	4.12	5.44
16	4.40	5.81
17	4.83	6.38
18	4.95	6.54
19	5.10	6.75
20	5.22	6.91
21	5.62	7.42

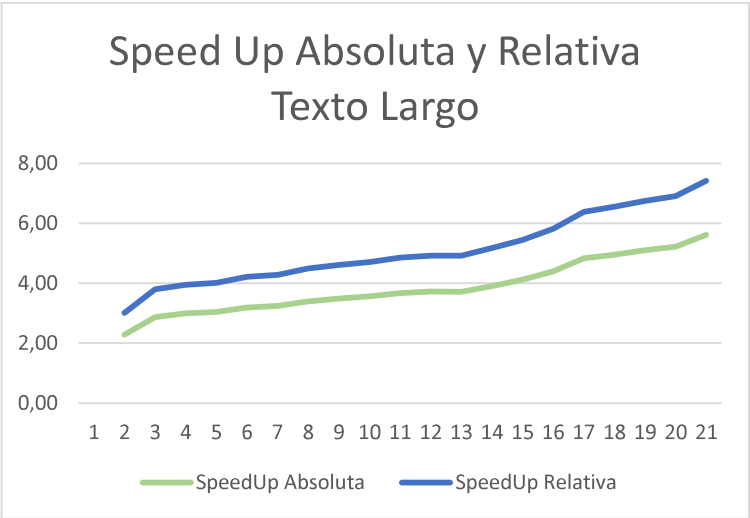


Figura 3.19. Speed-up absoluta y relativa en un clúster inalámbrico con texto largo.

La tabla 3.18 se muestran los valores de Speed-up absoluta y relativa cuando se ejecuta el algoritmo paralelo en un clúster inalámbrico para un texto corto, los valores oscila entre 2 y 6 speed-up absoluta y entre 3 y 7 relativa.

Tabla 3.18. Speed-up absoluta y relativa en un clúster inalámbrico con texto corto.

Texto Corto		
n	**Speed-up Absoluta**	**Speed-up Relativa**
2	2.23	2.81
3	2.80	3.51
4	2.92	3.67
5	3.04	3.82
6	3.16	3.97
7	3.21	4.04
8	3.37	4.23
9	3.41	4.28
10	3.58	4.50
11	3.66	4.60
12	3.70	4.65
13	3.81	4.79
14	3.88	4.87
15	4.17	5.24
16	4.37	5.49
17	4.77	6.00
18	4.83	6.07
19	4.97	6.24
20	5.35	6.73
21	5.40	6.79

En la Figura 3.20 se puede observar los valores de Tabla 3.18 en una gráfica donde las 2 curvas se mantienen en incremento durante todo el trayecto hasta llegar a n=21.

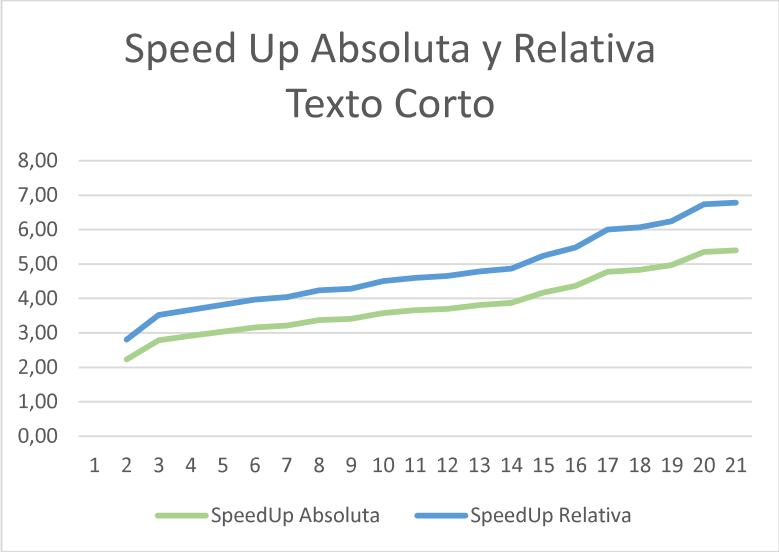


Figura 3.20. Speed-up absoluta y relativa en un clúster inalámbrico con texto corto.

En los ANEXOS A,B Y C se encuentra los valores que se utilizó para obtener speed-up absoluto y relativo en todos los escenarios presentados.

3.8. EFICIENCIA DEL ALGORITMO

Se ha realizado el cálculo de la eficiencia del algoritmo con los tiempos obtenidos, como se mencionó en la parte teórica, se define como el cociente entre la Speed-up y el número de trabajadores. Se ha tomado la Speed-up relativa como valor real, dado que se toma en cuenta el algoritmo serial normal más no el mejor algoritmo serial ya que ese no se podría paralelizar directamente.

Mediante los valores obtenidos se podrá analizar en qué caso se cumple adecuadamente la función de ataque por fuerza bruta del algoritmo S-DES para cada caso: paralelo local y paralelo en clúster (con texto corto y texto largo).

Se toma en cuenta los siguientes escenarios:

-Paralelo local (un PC)

-Paralelo en un clúster LAN

-Paralelo en un clúster inalámbrico

3.8.1. EFICIENCIA DEL ALGORITMO PARALELO EN UN PC

En la Tabla 3.19 se muestra la eficiencia al ejecutar el algoritmo serial vs paralelo en un solo equipo, para un texto largo y tomando en cuenta los trabajadores que actúan en el proceso. El decremento es constante en todo el intervalo que comprende 2 ,3 ,4 ,5 ,6 y 7 trabajadores.

Tabla 3.19. Eficiencia local en texto largo.

Texto Largo		
n	**Speed-up Relativa**	**Eficiencia**
2	1.68	0.84
3	1.77	0.59
4	1.91	0.48
5	1.79	0.36
6	1.85	0.31
7	1.77	0.25

La gráfica de eficiencia que se muestra en la Figura 3.21 permite una mejor comprensión de valores, aunque la gráfica es constante, no es lineal, sino que presenta ciertos puntos de interrupción en su forma, para 3 y 5 trabajadores. Si se analiza los valores a partir de 2 trabajadores existe una diferencia de 0.25 unidades hasta llegar a los 3, después la diferencia es menor, únicamente de 0.11 hasta llegar a 4, lo mismo sucede entre 4 y 5 trabajadores, la diferencia es de 0.12, mientras que de 5 a 6 es únicamente de 0.05.

Se muestra la gráfica de speed-up relativa y como eje secundario la eficiencia para poder ver hasta que punto la aplicación es escalable, por concepto para que sea escalable el valor debe ser superior a 0.5, lo que sucede cuando n=3.

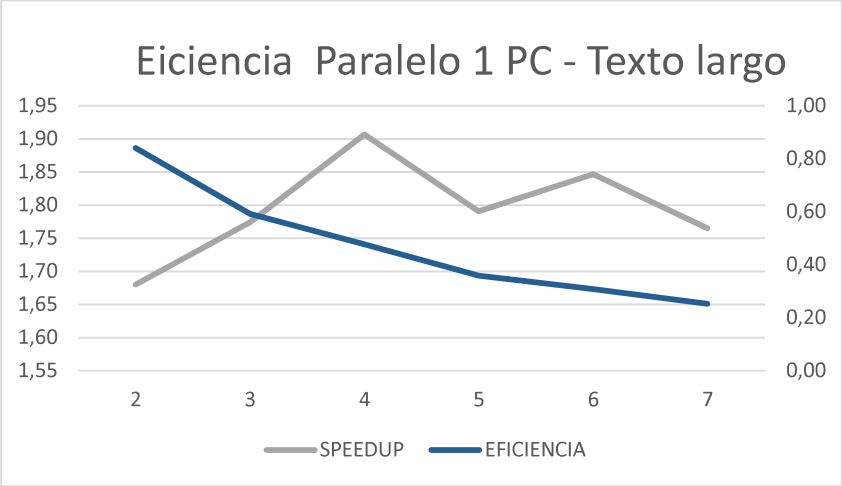


Figura 3.21. Eficiencia local en texto largo.

En la tabla 3.20 de describen los valores de eficiencia cuando se ejecuta el algoritmo de ataque secuencia vs paralelo en un solo equipo para un texto corto, tomando en cuenta el incremento de trabajadores de 2 a 7, los valores indican una mejora de eficiencia conforme los trabajadores aumentan, siendo el mejor valor 0.29 y el peor 0.80.

Tabla 3.20. Eficiencia local en texto corto.

Texto Corto		
n	**Speed-up Relativa**	**Eficiencia**
2	1.61	0.80
3	1.89	0.63
4	1.84	0.46
5	1.96	0.39
6	2.01	0.34
7	2.05	0.29

La Figura 3.22 plasma los valores generados en el cálculo de la eficiencia de la tabla 3.20, la mejora se ve con el decremento de la curva, dado que cuanto mejor sea el cómputo el valor será menor, una línea recta se observa entre el rango de 2 a 4 trabajadores es decir el decremento de 0.17 es constante, para 5, 6 y 7 trabajadores el decremento oscila entre 0.7 y 0.5, finalmente obteniendo la mejor eficiencia con el valor de 0.29 cuando actúan los 7 trabajadores.

Se mantiene el punto de escalaibilidad para en el valor n=3, cuando la eficiencia es superior a 0.5.

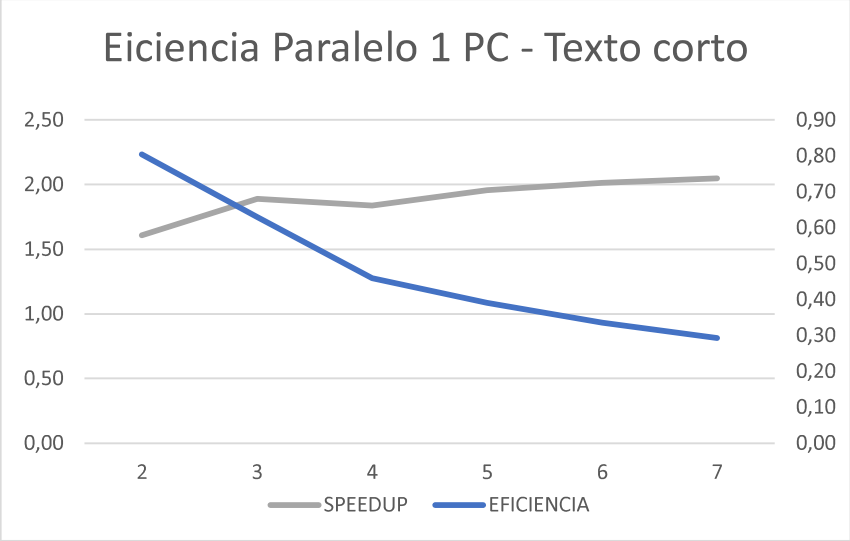


Figura 3.22. Eficiencia local en texto corto.

3.8.2. EFICIENCIA DEL ALGORITMO PARALELO EN UN CLÚSTER LAN

El mismo análisis se realiza para un clúster, considerando un texto largo, en la Tabla 3.21 se detallan los valores de eficiencia obtenidos, en el rango de 2 a 3 trabajadores se tienen valores superiores a 1 lo que indica un uso deficiente de recursos en la ejecución, ya que el valor máximo de la eficiencia es 1, por lo cual para la gráfica se considera a partir de 4 a 21 trabajadores.

En la Figura 3.23 se observa a detalle lo analizado en la tabla anterior, una curva decreciente con un pico en 12 por la distribución del trabajo en el cúster, a demás se ve que el punto en que n=8 el sistema es escalable.

Tabla 3.21. Eficiencia en el clúster LAN en texto largo.

Texto Largo		
n	**Speed-up Relativa**	**Eficiencia**
2	3.16	1.58
3	3.85	1.28
4	3.99	1.00
5	4.09	0.82
6	4.21	0.70
7	4.30	0.61
8	4.46	0.56
9	4.58	0.51
10	4.75	0.47
11	4.05	0.37
12	3.71	0.31
13	4.86	0.37
14	5.16	0.37
15	5.44	0.36
16	6.12	0.38
17	6.38	0.38
18	6.60	0.37
19	6.96	0.37
20	7.30	0.36
21	7.70	0.37

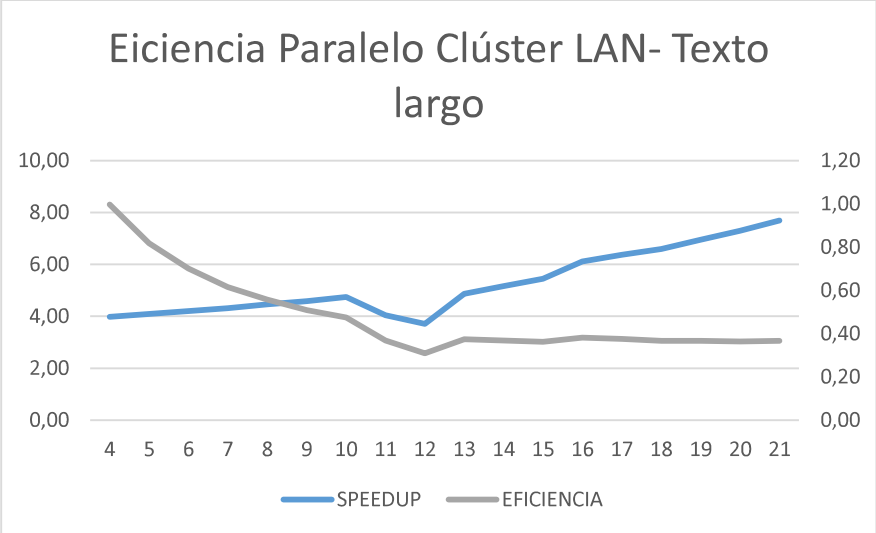


Figura 3.23. Eficiencia en un clúster LAN en texto largo.

Para el caso de ejecución del ataque con texto corto en un clúster LAN, se obtiene un resultado similar de decremento de valores tomando en cuenta apartir de n=4 con una eficiencia del 0.91, además se puede observar una constancia de eficiencias a partir de 17 trabajadores (Tabla 3.22).

Tabla 3.22. Eficiencia en el clúster LAN en texto corto.

Texto Corto		
n	**Speed-up Relativa**	**Eficiencia**
2	2.89	1.44
3	3.56	1.19
4	3.66	0.91
5	3.93	0.79
6	4.06	0.68
7	4.18	0.60
8	4.34	0.54
9	4.40	0.49
10	4.61	0.46
11	3.89	0.35
12	3.68	0.31
13	4.78	0.37
14	4.82	0.34
15	5.22	0.35
16	5.70	0.36
17	5.86	0.34
18	6.06	0.34
19	6.52	0.34
20	6.77	0.34
21	7.16	0.34

La gráfica de la Figura 3.24 muestra la mejora de la eficiencia del algoritmo cuando se hace uso de un clúster, siendo el mejor valor cuando el número de trabajadores totales trabaja; sin embargo, en n=8 el sistema es escalable.

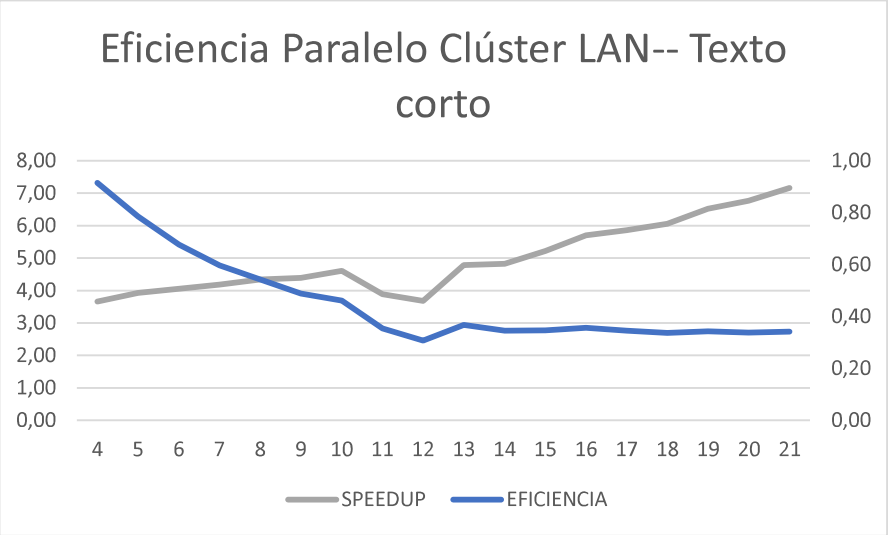


Figura 3.24. Eficiencia en un clúster LAN en texto corto.

3.8.3. EFICIENCIA DEL ALGORITMO PARALELO EN UN CLÚSTER INALÁMBRICO

El mismo análisis se realiza para un clúster inalámbrico, considerando un texto largo y corto, en la Tabla 3.23 y 24 respetivamente, se detallan los valores de eficiencia obtenidos, para ambos casos existe una disminuciòn constante hasta n=21 con 0.35 texto laro y 0.32 texto corto, sin picos en el transcurso de la gráfica, cosa que no sucedió en el cluster LAN por la distribución de trabajo.

En la figura 3.25 y 3.26 se observa el comportamiento de la curva de speed-up relativa con la eficiencia como eje secundarío, donde los valores fueron tomados desde n=4 partiendo de una eficiencia de 0.99.

El punto para que el sistema sea escalable es n=7, donde la eficiencia es mayor que 0.5 en el caso de un texto largo.

Tabla 3.23. Eficiencia en un clúster inalámbrico en texto largo.

Texto Largo		
n	**Speed-up Relativa**	**Eficiencia**
2	3.02	1.51
3	3.80	1.27
4	3.95	0.99
5	4.02	0.80
6	4.22	0.70
7	4.28	0.61
8	4.49	0.56
9	4.60	0.51
10	4.71	0.47
11	4.85	0.44
12	4.92	0.41
13	4.92	0.38
14	5.17	0.37
15	5.44	0.36
16	5.81	0.36
17	6.38	0.38
18	6.54	0.36
19	6.75	0.36
20	6.91	0.35
21	7.42	0.35

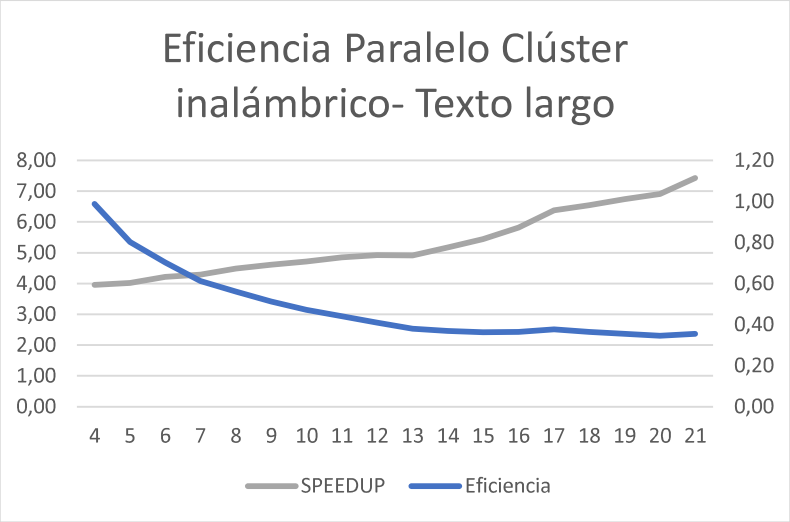


Figura 3.25. Eficiencia en un clúster inalámbrico en texto largo

Tabla 3.24. Eficiencia en un clúster inalámbrico en texto corto.

Texto Corto		
n	**Speed-up Relativa**	**Eficiencia**
2	2.81	1.40
3	3.51	1.17
4	3.67	0.92
5	3.82	0.76
6	3.97	0.66
7	4.04	0.58
8	4.23	0.53
9	4.28	0.48
10	4.50	0.45
11	4.60	0.42
12	4.65	0.39
13	4.79	0.37
14	4.87	0.35
15	5.24	0.35
16	5.49	0.34
17	6.00	0.35
18	6.07	0.34
19	6.24	0.33
20	6.73	0.34
21	6.79	0.32

El punto para que el sistema sea escalable es n=8, donde la eficiencia es mayor que 0.5 en el caso de un texto largo.

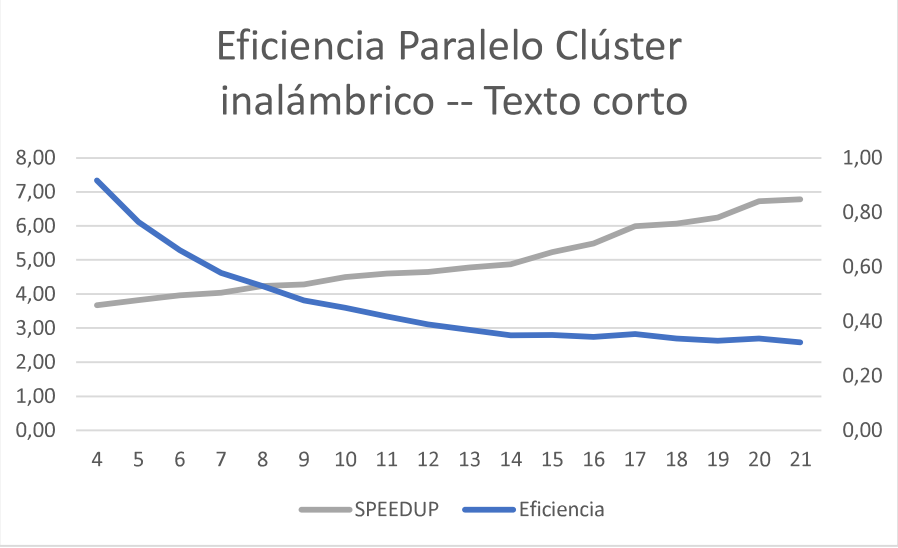


Figura 3.26. Eficiencia en un clúster inalámbrico en texto corto.

3.9. VENTAJAS DE LA COMPUTACIÓN PARALELA VS SECUENCIAL

Las ventaja principal de la computación paralela en este proyecto es la mejora de la eficiencia del algoritmo de ataque con respecto a los tiempos de ejecución consigueindo esto al dividir el problema en partes más pequeñas, mismas que fueron ejecutadas por cada trabajador configurado.

Algo adicional es el ahorro de recursos, es decir que no se tuvo que adquirir una supercomputadora sino que se agrupó un conjunto de equipos consiguiendo buenos resultados.

3.10. PORCENTAJE DE ESPACIO DE CLAVES

El porcentaje del espacio de claves que fue recorrido para el caso serial depende del orden en el que se ejecutó, mientras que en paralelo se comporta totalemte diferente.

Tomando la tabla 3.9 añadimos el campo %espacio para poder observar la diferencia (tabla 3.25) entre la ejecuión serial y paralela. En la parte serial observamos como los porcentajes suben conforme aumenta la dificultad de la clave, mientras que en paralelo local y clúster esto es independiente, a pesar de que los tiempos disminuyen el % de claves recorridas es grande, esto debido a que varios worker está trabajando a la vez para incrementar la eficiencia.

Figura 3.25. Porcentaje de espacio de claves reocrridas

Valor Binario	Serial (s)	% espacio	Paralelo local (s)	% espacio	Paralelo Clúster (s)	% espacio
"0000000001"	8.24	0.09	45.93	45.2	11.23	14.1
"0001100100"	15.89	9.7	43.82	68,1	12.01	25.3
"0011001000"	15.57	19.5	46.73	34.8	11.43	89.5
"0100101100"	30.72	29.3	47.5	16.4	11.97	76.4
"0110010000"	39.09	39.1	45.41	46.3	11.6	68.2
"0111110100"	46.02	48.8	46.16	77.21	12.6	39.1
"1001011000"	54.98	58.6	47.28	10.3	13.01	59.2
"1010111100"	60.92	68.3	47.05	65.7	12.8	71.1
"1100100000"	69.1	78.1	41.96	17.2	11.76	56.8
"1110000100"	77.59	87.9	42.04	67	11.34	71.5
"1111111111"	87	100	45.84	28.9	11.45	46.3

4. CONCLUSIONES Y RECOMENDACIONES

4.1. CONCLUSIONES

- Se diseñó el algoritmo paralelo para realizar el ataque por fuerza bruta de S-DES, tomando 2 situaciones: Reduciendo el tamaño del texto cifrado para evaluación de clave correcta y comandos que contiene la herramienta PTC.

- Se tomó tamaño fijo de texto cifrado independientemente de la longitud del texto original, ya que para evaluar si un texto tiene sentido, únicamente se tomó una línea de 50 caracteres, con esto se encontró la clave correcta y luego se procedió a descifrar la totalidad, ahorrando con esto un gran porcentaje de tiempo de ejecución, por ende, no se verá una diferencia grande en tiempo en el ataque de un archivo de gran tamaño versus uno de tamaño corto.

- Al analizar el funcionamiento de par for vs spmd se puede concluir que: ParFor es más eficiente ya que el algoritmo que internamente corre el comando distribuye de mejor manera los datos obteniendo mejores tiempos, cuando se utilizan entre 2 y 4 trabajadores los resultados para cada comando son parecidos, mientras que si se van aumentando se puede ver como SPMD demora más que PARFOR. SPMD necesita más especificaciones dentro del código ya que en un inicio todos los laboratorios resuelven el problema con todos los datos, condicionado el proceso se distribuye una carga igualitaria por lab; sin embargo, si lab ha obtenido la respuesta final la ejecución no puede detenerse hasta que todos los labs comuniquen que han finalizado, es decir que SPMD puede ser más eficiente si no fuera necesaria la comunicación entre *workers*, con PARFOR el código es sencillo, se utiliza como un bucle for normal, teniendo en cuenta que dentro de cada ciclo los valores se borran y por supuesto que el lazo no puede ser roto en media ejecución, los datos se toman de manera desordenada por lo que no sería factible utilizarlo para recursividad, la comunicación de trabajadores y en qué orden están procesando datos es transparente, hasta que se obtiene el valor esperado.

- Se pudo observar que, a más de la ejecución paralela en procesadores físicos, formó parte la concurrencia ya que los *workers* superaron en cantidades físicas los núcleos de los equipos, inicialmente si cada procesador realizara una ejecución dentro del clúster no podrían haber más de 8 trabajadores (2PC con 2 núcleos y 1 PC con 4); sin embargo, se trabajó con 21.

- Los tiempos promedios tomados para el cálculo de la speed-up absoluta y relativa, se tomaron modificando el tamaño de texto cifrado en ataque serial, de otra manera el tiempo de ejecución de un texto de 3 páginas tomaba alrededor de 2 horas y para el mejor ataque serial se tomó los tiempos del algoritmo sin generación de matriz de posibles claves.

- El resultado del speed-up absoluta cuando un texto largo cifrado es atacado por un solo equipo utilizando todos sus procesadores oscila entre 1.27 y 1.44, alcanzando su pico más alto cuando se ejecuta con 4 trabajadores del clúster local, a diferencia que, en texto corto, donde los rangos son: 1.28 a 1.63 alcanzado el valor más alto utilizando 7 *workers*.

- El resultado del speed-up absoluta cuando un texto largo cifrado es atacado por los 3 equipos que conforman el clúster oscila entre 2.39 y 5.82, valor máximo obtenido con 21 *workers*, lo mismo sucede con un texto corto, valor tope de 5.70 para el mismo número de *workers*.

- El resultado del speed-up relativa cuando un texto corto o largo cifrado es atacado por un solo equipo utilizando todos sus procesadores se mantiene la tendencia de speed-up absoluta, es decir el mejor valor de 1.91(texto largo) para 4 *workers* y 2.05(texto corto) para 7 *workers*. Lo mismo ocurre para el caso de clúster: 7.70 y 7.16 para texto corto y largo respectivamente, cuando se ejecuta con 21 *workers*.

- Dados los resultados no se puede concluir que a mayor número de *workers* mayor aceleración (speed-up) dado que esto depende de cómo el comando parfor distribuya el trabajo dentro de los *workers* participantes en la ejecución de la aplicación, aunque la tendencia en la mayoría de los casos apuntó a obtener mejores valores con la utilización de todos los *workers*.

- Con los valores de eficiencia para cada caso antes mencionado se puede observar que el paralelismo en un solo equipo va mejorando conforme aumente los *workers* en la ejecución hasta llegar a 7, mientras que en un clúster mejora hasta llegar a 12 *workers*, desde ese punto se mantiene algo constante hasta llegar a 21 *workers*, el aumento se da, pero no es notorio, esto podría variar dependiendo de las distribuciones de datos dentro de los trabajadores.

- Respecto a resultados finales se puede concluir que con un solo equipo utilizando sus capacidades computaciones de manera serial, tomando como referencia la ejecución de un algoritmo de ataque por fuerza bruta al S-DES, con un texto de

entrada de alrededor de 3 páginas se obtiene un promedio de 87.83 s, cuando se utilizan todos los procesadores del equipo se tiene un tiempo promedio de 48.34 s y el mejor tiempo se encontró en la ejecución de 7 *workers*: 45.8 , mientras que utilizando el clúster con 3 equipos LAN se obtienen un valor promedio de 18.34, y el mejor valor encontrado con la ejecución de 21 *workers* : 11.41, finalmente para un clúster inalámbrico el valor promedio es:18.09 y el mejor es:11.83.

- La mejora en porcentaje del paralelismo aplicado en el presente proyecto considerando los mejores tiempos de ejecución en paralelo y el tiempo promedio de varias ejecuciones seriales tiene el valor de 52.14%, esto dando una respuesta favorable cuando se utilizan gran cantidad de datos.

- A pesar que los tiempos bajaron conforme el número de *workers* incrementaba no se puede concluir que el sistema es escalable cuando se utiliza la totalidad de los recursos del clúster, ya que la eficiencia no se mantuvo constante y superior a 0.5 en todo el escenario, en paralelo local se obtuvo el sistema escalable con 4 *workers*, mientras que para el clúster LAN o inalámbrico con 7 a 8 *workers*.

4.2. RECOMENDACIONES

- Utilizar la misma versión del software MATLAB en todos los PCs que conforman el clúster para que no exista incompatibilidad con las versiones de los toolboxes requeridos,

- Hacer que las funciones o líneas de código a paralelizar sean independientes entre si, es decir que un resultado dependa de otro ya que de esta manera no se puede trabajar con paralelismo,

- Si no se utilizan demasidos datos o el problema no consume un alto procesamiento, verificar si el paralelismo da buenos resultados ya que puede no ser necesario, por esto es necesario realizar una comparación de tiempos en cada sección que se desee paralelizar.

- Utilizar de preferencia una red cableada ya que la red inalámbrica puede tener intermitencias y esto causa desconexión de los trabajadores del cluster, por tanto se obtendrá un error en la ejecución.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] H. Huerta, *Hackeo mediante Ataque de fuerza Bruta,* Mexico , 2016 .

[2] M. d. C. R. Ternero, *El Modelo de referencia OSI (ISO 7498),* Sevilla, 2005.

[3] D. Bishop, Introduction to Cryptography with Java Applets, Massachusetts: Jones and Bartlett Publishers, Inc., 2003.

[4] E. F. Schaefe, «A SIMPLIFIED DATA ENCRYPTION STANDARD ALGORITHM,» *CRIPTOLOGIA,* pp. 77-86, 1996.

[5] Julián, «BLOG: IDEA SECUNDARIA,» 03 Febrero 2012. [En línea]. Available: https://ideasecundaria.blogspot.com/2012/02/las-armas-las-carga-el-diablo-y-este.html. [Último acceso: 13 10 2018].

[6] O. Long, «CCM.NET,» 19 06 2018. [En línea]. Available: https://ccm.net/contents/134-introduction-to-encryption-with-des. [Último acceso: 12 10 2018].

[7] C. M. Curtin, «interhack.net,» 08 02 2007. [En línea]. Available: http://www.interhack.net/projects/deschall/. [Último acceso: 13 10 2018].

[8] J. Gilmore, «Cracking DES, Editor,» The Electronic Frontier Foundation, San Francisco, 1996.

[9] D. Baker, «US GOVERNMENT'S ENCRYPTION STANDARD BROKEN IN LESS THAN A DAY,» E.E.U.U, 1999.

[10] D. Hankerson, G. Hoffman, L. D.A., C. C. Lindner, P. K.T., R. C.A. y J. Wall, Coding Theory and Cryptography: The Essentials, U.S.A: Taylor and Francis Group, 2002.

[11] S. R. S. Sandoval Acosta, «LA CRIPTOGRAFÍA,» IQUITOS, 2014.

[12] J. Arteaga y I. García, «TECNOLOGÍAS DE ENCRIPTACIÓN DE INFORMACIÓN Y PROTOCOLOS SEGUROS,» Ibarra, 2003.

[13] A. Fúster Sabater, «Criptografía de Clave secreta: Cifrado en flujo,» Madrid, 2009.

[14] V. Eijkhout, «Introduction to Parallel Computing,» Texas, 2011.

[15] S. Gill, «Parallel Programming,» *The Computer Journal,* vol. I, nº 1, pp. 2-10, 1958.

[16] J. L. Aguilar Castro y E. Leiss, Introducción a la Computación Paralela, Mérida, 2004.

[17] J. Perez Mato, Supercomputadores Historia y Actualidad, Las Palmas de Gran Canaria, 2009.

[18] F. Almeida, D. Giménez, J. M. Mantas y A. M. Vidal, «Sobre la situación del paralelismo y la programación paralela en los Grados de Ingeniería Informática,» *ReVisión,* vol. III, nº 1, 2010.

[19] D. Bader y R. Pennington, «Cluster Computing: Applications,» *The International Journal of High Performance Computing Applications,* vol. XV, nº 2, pp. 181-185, 2001.

[20] N. Sadashiv y S. M. D. Kumar, «Cluster, Grid and Cloud Computing: A Detailed Comparison Computer Science & Education (ICCSE 2011),» de *The 6th International Conference on*, Singapore, 2011.

[21] J. Peña, «Middlewares para Sistemas de Alto,» 8 Agosto 2017. [En línea]. Available: http://laurel.datsi.fi.upm.es/_media/docencia/asignaturas/ccg/middlewares-ccg.pdf. [Último acceso: 24 11 2018].

[22] I. Gomez, «Clusters de Alto Rendimiento,» Guadalajara, 2009.

[23] J. Gray y D. P. Siewiorek, «High Availability Computer Systems,» *IEEE Computer Magazine Draft,* vol. XIIII, nº 9, pp. 39-48 , 1991.

[24] P. Ghildiyal, «pawangh.blogspot.com,» 01 06 2014. [En línea]. Available: http://pawangh.blogspot.com/2014/05/mpi-vs-openmp.html. [Último acceso: 31 10 2018].

[25] M. Alcubierre, «Introducción a FORTRAN,» Ciudad de México, 2005.

[26] G. Fox y W. Furmanski, «Java for Parallel Computing and as a General Language for Scientiand Engineering Simulation and Modelling,» New York, 1997.

[27] M. Püschel, «How to Write Fast Numerical Code,» Spring, 2012.

[28] L. Müller, «Evaluación del rendimiento de Algoritmos Paralelos y/o Concurrentes,» Asución, 2011.

[29] S. Cubas, «Arquitectura de Busqueda Basada en Tecnicas Soft Computing para la Resolucion de Problemas Combinatorios en Diferentes Dominios de Aplicacion,» Valencia, 2010.

[30] F. Bris, «CÓMO MEDIR LA EFICIENCIA EN ALGORITMOS PARALELOS,» Linares, 2011.

[31] T. G. Lewis y H. El-Rewini, Introduction to Parallel Computing, New Jersey: Prentice-Hall, 1992.

[32] A. (. d. Kriptópolis), «Frecuencia de las letras en castellano: "La Regenta",» 16 12 2013. [En línea]. Available: https://web.archive.org/web/20131216055136/http://www.kriptopolis.org/frecuencia-letras-castellano. [Último acceso: 12 12 2018].

[33] J. Nino, «ANÁLISIS Y REPRESENTACIÓN DE FRECUENCIAS DE LETRAS EN EL LIBRO “EL INGENIOSO HIDALGO DON QUIJOTE DE LA MANCHA” CON POWERSHELL,» 26 01 2017. [En línea]. Available: https://www.jesusninoc.com/01/26/analisis-y-representacion-de-frecuencias-de-letras-en-el-libro-el-ingenioso-hidalgo-don-quijote-de-la-mancha-con-powershell-parte-7/. [Último acceso: 10 01 2019].

6. ANEXOS

ANEXO A: Speed-up Absoluta y relativa paralelo local

Texto Largo			
n	tn (s)	ts(s)	Speed-up Absoluta
2	52.295	66.44	1.27
3	49.51	66.44	1.34
4	46.07	66.44	1.44
5	49.06	66.44	1.35
6	47.565	66.44	1.40
7	49.76	66.44	1.34

Texto Largo			
n	tn (s)	ts(s)	Speed-up Relativa
2	52.295	87.83	1.68
3	49.51	87.83	1.77
4	46.07	87.83	1.91
5	49.06	87.83	1.79
6	47.565	87.83	1.85
7	49.76	87.83	1.77

Texto Corto			
n	tn (s)	ts(s)	Speed-up Absoluta
2	49.745	63.65	1.28
3	42.35	63.65	1.50
4	43.505	63.65	1.46
5	40.875	63.65	1.56
6	39.735	63.65	1.60
7	39.055	63.65	1.63

Texto Corto			
n	tn (s)	ts(s)	Speed-up Relativa
2	49.745	80	1.61
3	42.35	80	1.89
4	43.505	80	1.84
5	40.875	80	1.96
6	39.735	80	2.01
7	39.055	80	2.05

ANEXO B: Speed-up Absoluta y relativa paralelo clúster LAN

Texto Largo			
n	tn (s)	ts(s)	Speed-up Absoluta
2	27.8	66.44	2.39
3	22.81	66.44	2.91
4	22.02	66.44	3.02
5	21.49	66.44	3.09
6	20.86	66.44	3.18
7	20.40	66.44	3.26
8	19.68	66.44	3.38
9	19.16	66.44	3.47
10	18.50	66.44	3.59
11	21.71	66.44	3.06
12	23.68	66.44	2.81
13	18.06	66.44	3.68
14	17.01	66.44	3.91
15	16.14	66.44	4.12
16	14.36	66.44	4.63
17	13.77	66.44	4.83
18	13.31	66.44	4.99
19	12.62	66.44	5.26
20	12.04	66.44	5.52
21	11.41	66.44	5.82

Texto Largo			
n	tn (s)	ts(s)	Speed-up Relativa
2	27.8	87.83	3.16
3	22.81	87.83	3.85
4	22.02	87.83	3.99
5	21.49	87.83	4.09
6	20.86	87.83	4.21
7	20.40	87.83	4.30
8	19.68	87.83	4.46
9	19.16	87.83	4.58
10	18.50	87.83	4.75
11	21.71	87.83	4.05
12	23.68	87.83	3.71
13	18.06	87.83	4.86
14	17.01	87.83	5.16
15	16.14	87.83	5.44
16	14.36	87.83	6.12
17	13.77	87.83	6.38
18	13.31	87.83	6.60
19	12.62	87.83	6.96
20	12.04	87.83	7.30
21	11.41	87.83	7.70

Texto Corto			
n	tn (s)	ts(s)	Speed-up Absoluta
2	27.7	63.65	2.30
3	22.47	63.65	2.83
4	21.86	63.65	2.91
5	20.36	63.65	3.13
6	19.72	63.65	3.23
7	19.14	63.65	3.33
8	18.42	63.65	3.46
9	18.19	63.65	3.50
10	17.37	63.65	3.66
11	20.55	63.65	3.10
12	21.73	63.65	2.93
13	16.73	63.65	3.80
14	16.58	63.65	3.84
15	15.34	63.65	4.15
16	14.04	63.65	4.53
17	13.66	63.65	4.66
18	13.21	63.65	4.82
19	12.28	63.65	5.18
20	11.82	63.65	5.39
21	11.17	63.65	5.70

Texto Corto			
n	tn (s)	ts(s)	Speed-up Relativa
2	27.7	80	2.89
3	22.47	80	3.56
4	21.86	80	3.66
5	20.36	80	3.93
6	19.72	80	4.06
7	19.14	80	4.18
8	18.42	80	4.34
9	18.19	80	4.40
10	17.37	80	4.61
11	20.55	80	3.89
12	21.73	80	3.68
13	16.73	80	4.78
14	16.58	80	4.82
15	15.34	80	5.22
16	14.04	80	5.70
17	13.66	80	5.86
18	13.21	80	6.06
19	12.28	80	6.52
20	11.82	80	6.77
21	11.17	80	7.16

ANEXO C: Speed-up Absoluta y relativa paralelo clúster inalámbrico

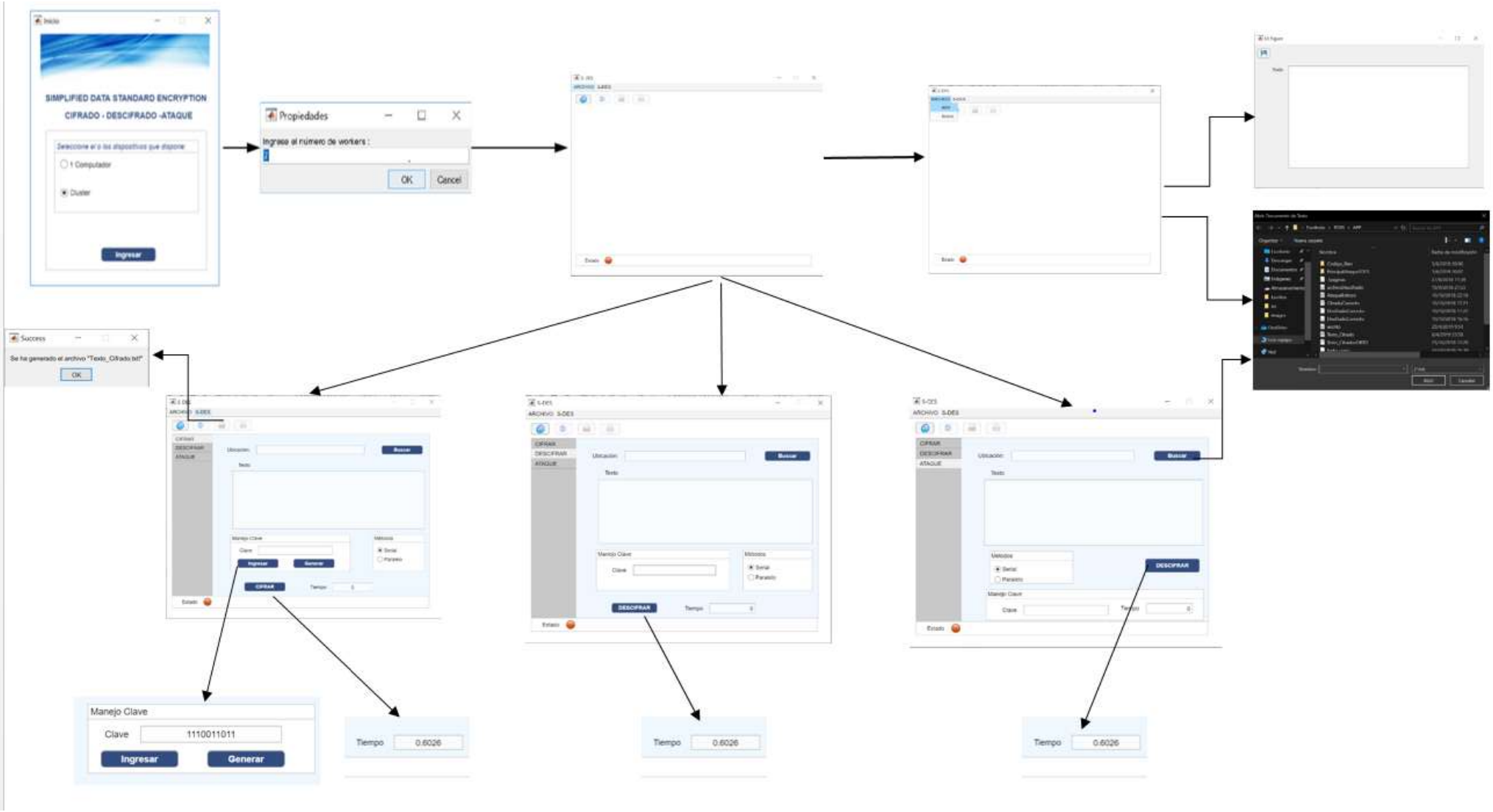
Texto Largo			
n	tn (s)	ts(s)	Speed-up Absoluta
2	29.11	66.44	2.28
3	23.12	66.44	2.87
4	22.24	66.44	2.99
5	21.87	66.44	3.04
6	20.81	66.44	3.19
7	20.51	66.44	3.24
8	19.58	66.44	3.39
9	19.07	66.44	3.48
10	18.64	66.44	3.56
11	18.10	66.44	3.67
12	17.85	66.44	3.72
13	17.86	66.44	3.72
14	16.99	66.44	3.91
15	16.14	66.44	4.12
16	15.11	66.44	4.40
17	13.77	66.44	4.83
18	13.42	66.44	4.95
19	13.02	66.44	5.10
20	12.72	66.44	5.22
21	11.83	66.44	5.62

Texto Largo			
n	tn (s)	ts(s)	Speed-up Relativa
2	29.11	87.83	3.02
3	23.12	87.83	3.80
4	22.24	87.83	3.95
5	21.87	87.83	4.02
6	20.81	87.83	4.22
7	20.51	87.83	4.28
8	19.58	87.83	4.49
9	19.07	87.83	4.60
10	18.64	87.83	4.71
11	18.10	87.83	4.85
12	17.85	87.83	4.92
13	17.86	87.83	4.92
14	16.99	87.83	5.17
15	16.14	87.83	5.44
16	15.11	87.83	5.81
17	13.77	87.83	6.38
18	13.42	87.83	6.54
19	13.02	87.83	6.75
20	12.72	87.83	6.91
21	11.83	87.83	7.42

Texto Corto			
n	tn (s)	ts(s)	Speed-up Absoluta
2	28.51	63.65	2.23
3	22.77	63.65	2.80
4	21.81	63.65	2.92
5	20.94	63.65	3.04
6	20.17	63.65	3.16
7	19.81	63.65	3.21
8	18.89	63.65	3.37
9	18.67	63.65	3.41
10	17.77	63.65	3.58
11	17.40	63.65	3.66
12	17.19	63.65	3.70
13	16.72	63.65	3.81
14	16.41	63.65	3.88
15	15.27	63.65	4.17
16	14.58	63.65	4.37
17	13.33	63.65	4.77
18	13.18	63.65	4.83
19	12.82	63.65	4.97
20	11.89	63.65	5.35
21	11.79	63.65	5.40

Texto Corto			
n	tn (s)	ts(s)	Speed-up Relativa
2	28.51	80	2.81
3	22.77	80	3.51
4	21.81	80	3.67
5	20.94	80	3.82
6	20.17	80	3.97
7	19.81	80	4.04
8	18.89	80	4.23
9	18.67	80	4.28
10	17.77	80	4.50
11	17.40	80	4.60
12	17.19	80	4.65
13	16.72	80	4.79
14	16.41	80	4.87
15	15.27	80	5.24
16	14.58	80	5.49
17	13.33	80	6.00
18	13.18	80	6.07
19	12.82	80	6.24
20	11.89	80	6.73
21	11.79	80	6.79

ANEXO D: DIAGRAMA DE SECUNCIA DE CONTROLES



ORDEN DE EMPASTADO