Universidad Del Valle de Guatemala

Computación Paralela y Distribuida

Ing. Luis García

PROYECTO 2

Javier Prado, 21486

Brayan España,

Cesar López, 22535

Guatemala, 31 de Octubre de 2025

Parte A

Investigación Sobre DES

Data Encryption Standard es un algoritmo de cifrado simetrico de bloques desarrollado para proteger información mediante una clave compartida, hoy en día ya no es recomendable usarlo ya que no es muy seguro frente a ataques modernos, (clave de 56 bits es muy corta). Opera sobre bloques de 64 bits y utiliza una clave nominal de 64 bits de la cual 8 bits son de paridad, por lo que la clave efectiva tiene 56 bits. DES es un esquema del tipo Feistel: divide cada bloque en dos mitades y aplica 16 rondas idénticas de transformación que mezclan la mitad derecha con una función no lineal dependiente de la subclave de la ronda.

Las transformaciones internas combinan permutaciones fijas, una expansión de 32 a 48 bits, operaciones XOR con subclaves, cajas S (S-boxes) que introducen no linealidad (convirtiendo 6 bits en 4) y una permutación P final. Todo ello está diseñado para difundir y confundir los bits del bloque original de forma determinista y reversible.

Cifrado:

Para cifrar un bloque con DES primero se genera el conjunto de 16 subclaves de 48 bits a partir de la clave de 56 bits mediante el llamado key-schedule: se aplica una permutación inicial PC-1 que elimina los bits de paridad, se divide la clave en dos mitades de 28 bits y en cada ronda se realizan rotaciones (shifts) d eesas mitades segun una tabla de desplazamientos. Las mitades rotadas se recombinan y se aplica PC-2 para producir la subclave de la ronda. Con las subclaves listas, el bloque de 64 bits sufre la Permutación Inicial (IP) y se separa en L0 y R0 (de 32 bits cada uno). A lo largo de 16 rondas Feistel se calcula:

Li = R_{i-1}

R_i = L_{I-1}

XOR F(R_{i-1}, K_i)

Donde la función F expande R_{i-1}a 48 bits, la mezcla XOR con la subclave Ki, pasa cada trozo de 6 bits por su correspondiente S-box para obtener 32 bits y aplica la permutación P, al terminar las 16 rondas se realiza el swap final (concatenar R16|L16) y se aplica la permutación inversa IP-1 para obtener el texto cifrado. En la práctica hay que considerar además cómo se manejan bloques múltiples (modo de operación: ECB, CBC, etc.), el padding cuando el mensaje no es multiplo de 8 bytes y la correcta gestión de orden bits/bytes para que la implementación sea compatible con vectores de prueba conocidos.

Descifrado:

El descifrado con DES utiliza exactamente el mismo mecanimos interno que el cifrado, la diferencia crucial radica en el orden de aplicación de las subclaves: para recuperar el texto original se aplican las subclaves en orden inverso (K16, K15, ..., K1) dentro del mismo bucle de 16 rondas Festel. Es decir, tras aplicar IP y dividir en L0 y R0, se realizan las 16 rondas usando las subclaves en orden inverso. La simetría de la estructura Feistel garantiza que esta secuencia revierta las transformaciones efectuadas durante el cifrado, y al aplicar permutación final IP-1 tras el último paso se recupera el bloque de 64 bits original (o el bloque con padding, si se aplicó). Desde la perspectiva de implementación y pruebas, es fundamental validar que la generación de subclaves (rotaciones y permutaciones PC-1/PC-2) y las S-Boxes esten correctas, ya que un error en cualquiera de estos pasos rompe la reversibilidad, por eso se usan vectores de prueba estándar y se verifica además que el modo de operacion y el padding coincidan entre cifrado y descifrado.

Diagrama de Flujo Algoritmo DES

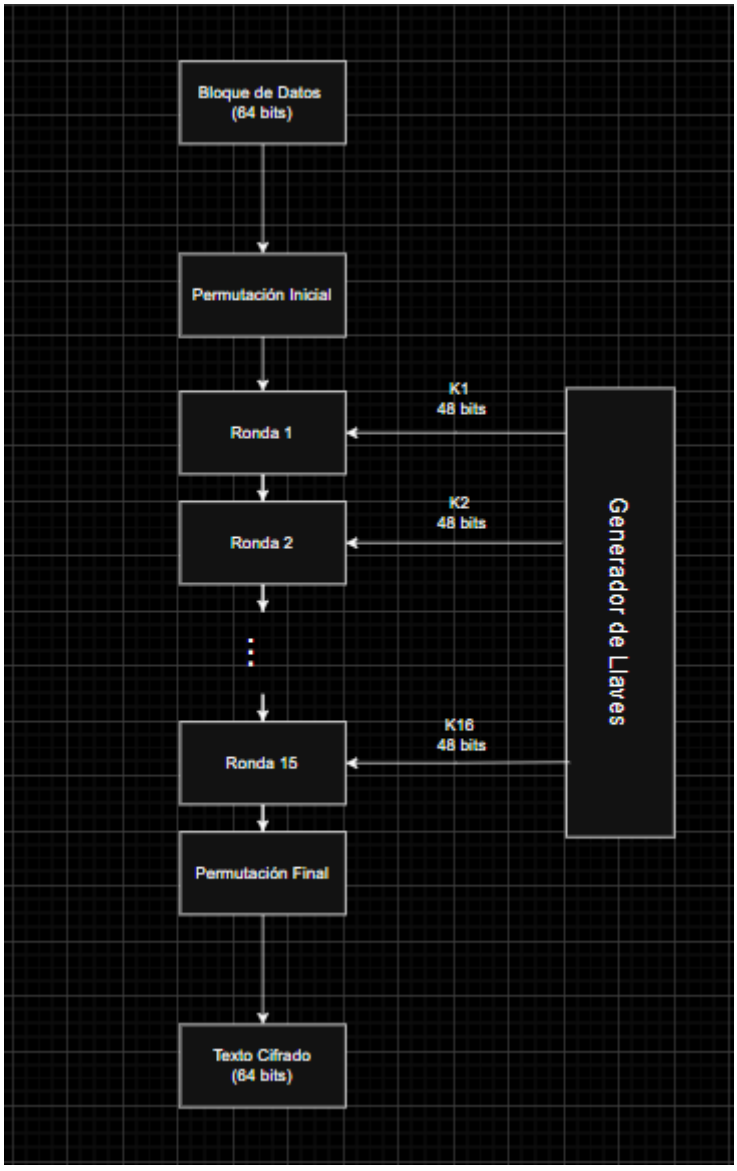

Diagrama de Generador de Llaves

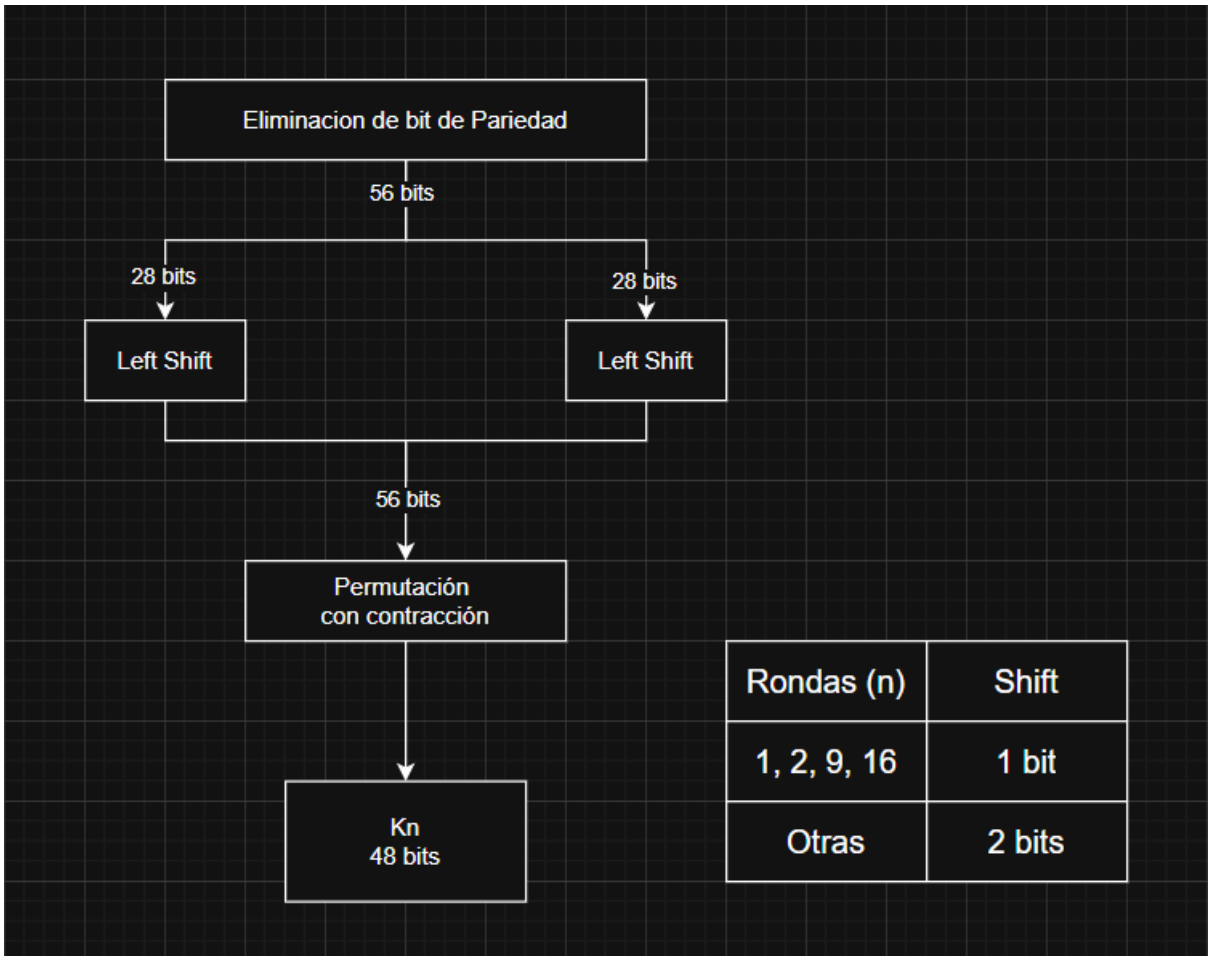

Diagrama por cada Round:

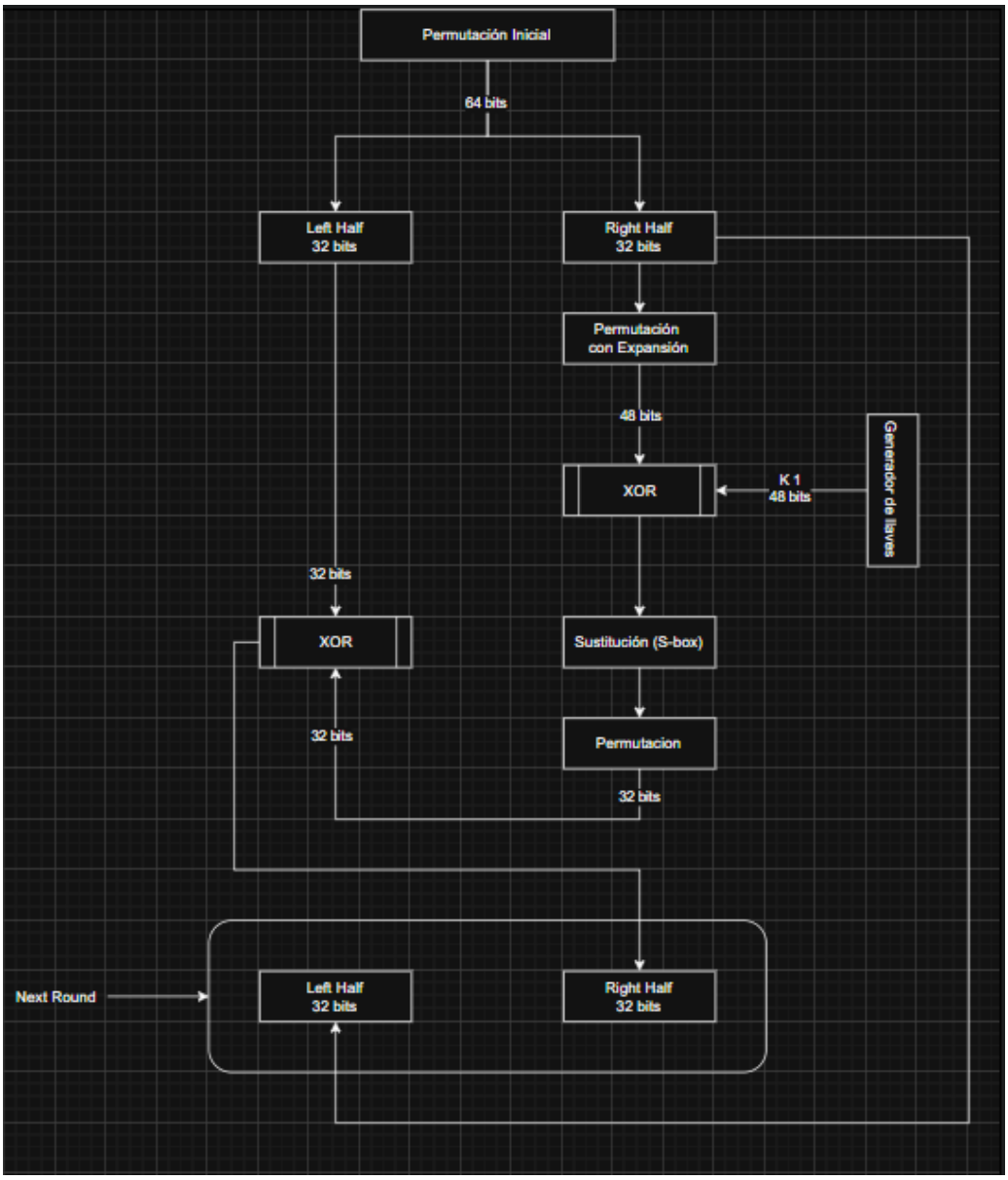

Diagrama Rutina decrypt (decrypt_u64)

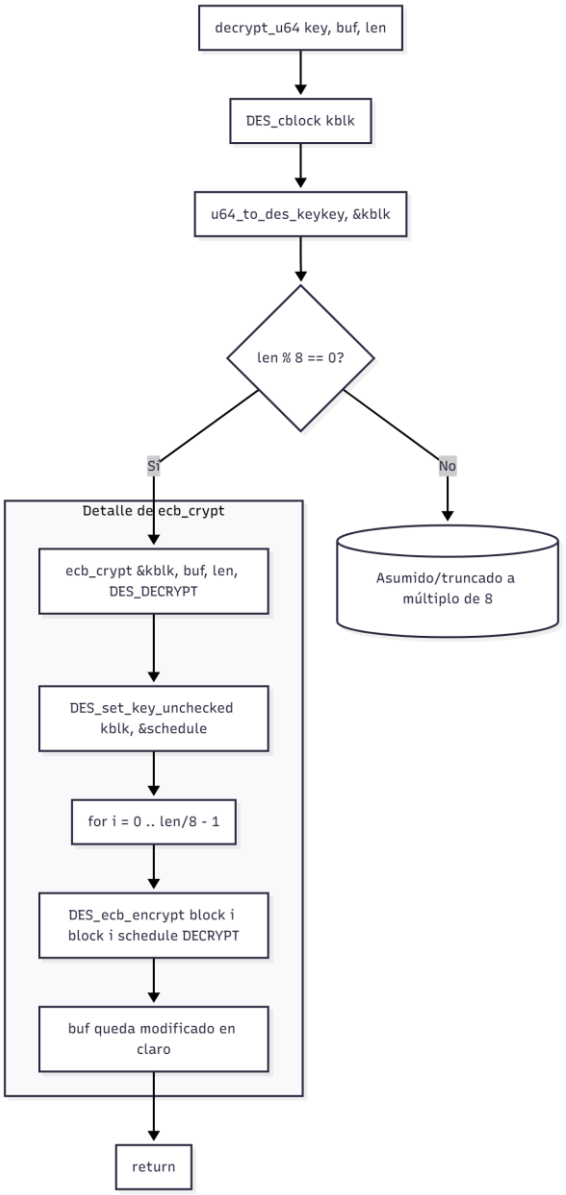

Diagrama Rutina encrypt (ecb_crypt)

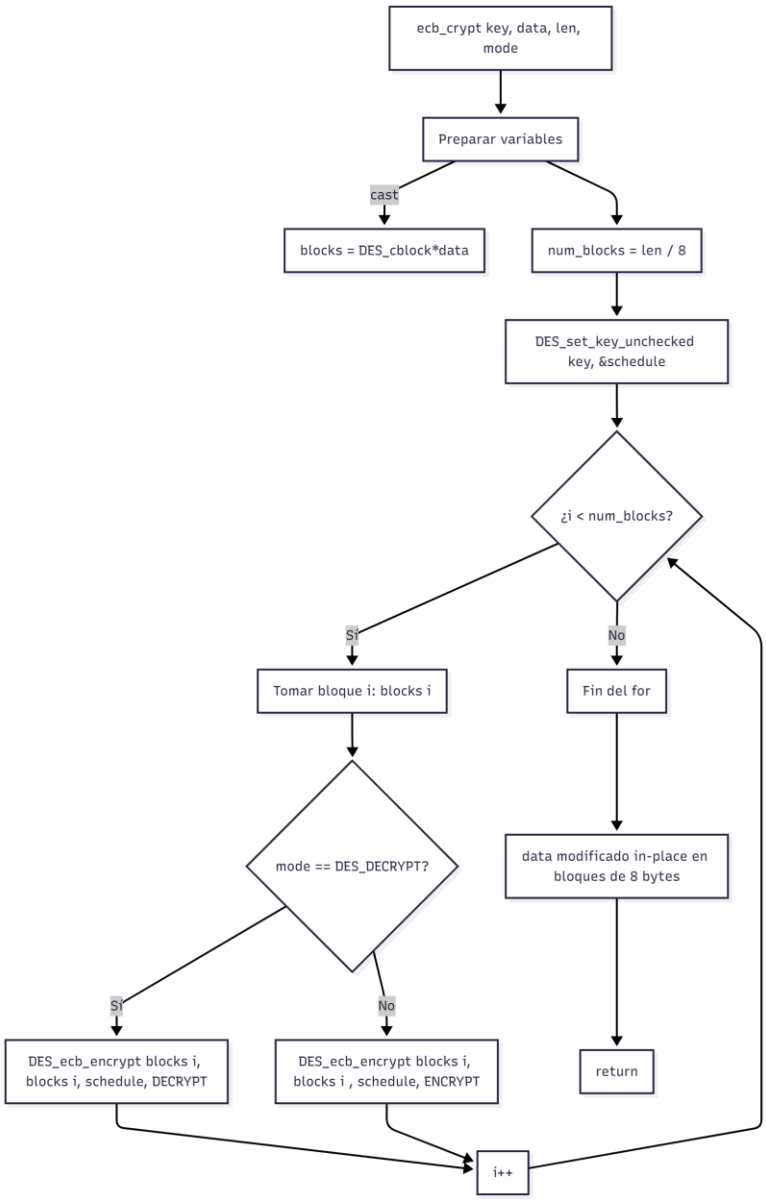

Diagrama Rutina tryKey

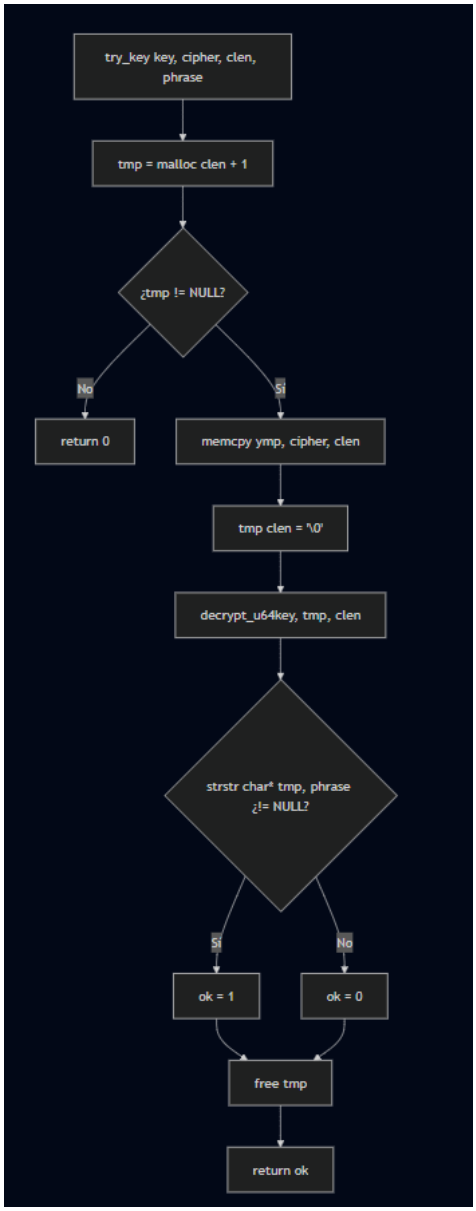

Diagrama Rutina memcpy

La funcion de memcpy (MEMORY COPY) se utiliza para copiar un úmero especifico de bytes de la memoria origen a la memoria destino. En el cual con los argumentos (destino, origen, tamaño) copia byte por byte, desde el principio hasta el final, son considerar el contenido de los bytes. Y devuelve el puntero a la memoria de destino.

Diagrama Rutina srcsrc

La funcion strstr (String String) se utiliza para encontrar la primera aparición de una subcadena dentro de una cadena más grande. En el cual recorre la cadena principal carácter por carácter, intenando coincidir completamente con la subcadena a buscar. Tan pronto encuentra una secuancia de caracteres a la subcadena, se detiene. Si se usa strstr("probando clave secreta", "clave"), la función devuelve un puntero que apunta a la 'c' de "clave" dentro de la cadena principal.

Describa y explique el uso y flujo de comunicación de las primitivas de MPI:
a. MPI_Irecv
b. MPI_Send
c. MPI_Wait

En MPI, MPI_Irecv, MPI_Send y MPI_Wait se usan juntos para implementar un flujo de comunicación eficiente y sin bloqueos. El receptor (el rank 0) postea una recepción no bloqueante con **MPI_Irecv**(buf, count, type, source, tag, comm, &req): la llamada retorna de inmediato y deja un request "en vuelo" indicando que está listo para recibir un mensaje que case en comunicador, fuente y tag. Mientras tanto, cualquier emisor (un worker que encuentra la llave) envía el dato con **MPI_Send**(buf, count, type, dest, tag, comm). Cuando el mensaje llega, el receptor completa la operación llamando **MPI_Wait(**&req, &status), que bloquea hasta que la recepción finalice y garantiza que el buffer de destino contiene el mensaje íntegro. Este patrón permite que el receptor continúe computando entre la publicación de la recepción (Irecv) y su finalización (Wait), evitando esperas activas o bloqueos innecesarios.

Parte B

1. **Ya que estamos familiarizados con el proyecto, vamos a analizar el problema más a fondo. Modifique su programa para que cifre un texto cargado desde un archivo (.txt) usando una llave privada arbitraria (como parámetro). Muestra una captura de pantalla evidenciando que puede cifrar y descifrar un texto sencillo (una oración) con una clave sencilla (por ejemplo: 42).**

```
javilejo@LAPTOP-OMB9KRA6:/mnt/c/Users/javil/OneDrive/Documentos/U/Computacion Paralela/P2_CParalela$ ./build/encrypt_linux files/plain.txt files/cipher
_42.bin 42
OK. Generado files/cipher_42.bin con llave decimal 42
javilejo@LAPTOP-OMB9KRA6:/mnt/c/Users/javil/OneDrive/Documentos/U/Computacion Paralela/P2_CParalela$ mpirun --oversubscribe -np 4 ./build/bruteforce fi
les/cipher_42.bin "es una prueba" 6
[rank 0] probando k=0 (rango [0, 16))
RESULT: KEY FOUND
Key     : 21
Plain   : Esta es una prueba de proyecto 2


Time(s) : 0.000053
[rank 1] probando k=16 (rango [16, 32))
[rank 2] probando k=32 (rango [32, 48))
[rank 3] probando k=48 (rango [48, 64))
```

2. **Una vez listo el paso anterior, proceder a hacer las siguientes pruebas, evidenciando todo en su reporte. Para todas ellas utilice 4 procesos (-np 4). El texto que cifrar/descifrar: "Esta es una prueba de proyecto 2". La palabra clave a buscar es: "es una prueba de":**
    a. **Mida el tiempo de ejecución en romper el código usando la llave 123456L**

```
javilejo@LAPTOP-OMB9KRA6:/mnt/c/Users/javil/OneDrive/Documentos/U/Computacion Paralela/P2_CParalela$ mpirun --oversu
bscribe -np 4 ./build/bruteforce files/cipher_123456.bin "es una prueba de" 17
[rank 0] probando k=0 (rango [0, 32768))
[rank 1] probando k=32768 (rango [32768, 65536))
[rank 2] probando k=65536 (rango [65536, 98304))
[rank 3] probando k=98304 (rango [98304, 131072))
RESULT: KEY FOUND
Key     : 14496
Plain   : Esta es una prueba de proyecto 2


Time(s) : 0.013521
Tiempo de ejecucion del algoritmo : 0.013521
```

b. Mida el tiempo de ejecución en romper el código usando la llave: (256/4). Es decir, 18014398509481983L. [spoiler: se tardará mucho, si es que termina, no se ofusquen si no termina].
   el programa no terminó en un tiempo razonable.
c. Mida el tiempo de ejecución en romper el código usando la llave: (256/4)+1. Es decir, 18014398509481984L.
   Tampoco termina, la posición de la llave es muy alta, por lo cual se tardaría mucho en encontrar la llave por ser tiempo exponencial.

d. Reflexione lo observado y el comportamiento del tiempo en función de la llave.
   Con el método de fuerza bruta, el tiempo para encontrar la llave crece linealmente con su posición en el rango de búsqueda. Si probamos desde 0 hacia arriba, una llave pequeña cae pronto, una llave cerca de 2^54 tardaría mucho. Por lo tanto, el bruteforce sobre 2^54 llaves es inviable.

3. **Como podemos ver, el acercamiento "naive" no es el mejor posible. Proponga, analice, e implemente 2 alternativas al acercamiento "naive". Tenga como objetivo en mente encontrar un algoritmo que tenga mejor "tiempo paralelo esperado".**
4. **Describa los acercamientos propuestos, puede apoyar con diagramas de flujo, pseudocódigo, o algoritmo descriptivo.**

Algoritmo Master–Worker con *chunking* dinámico + orden aleatorio de llaves + paro temprano

El programa implementa un ataque de fuerza bruta paralelo para romper una clave DES (Data Encryption Standard) utilizando el modelo Master–Worker con MPI.
Su objetivo es probar distintas claves de 56 bits hasta encontrar aquella que, al descifrar un texto cifrado, contenga una palabra clave conocida.

El algoritmo divide el espacio total de claves entre múltiples procesos. Un proceso actúa como maestro (rank 0) y los demás como trabajadores (workers). El maestro coordina la distribución de trabajo y los trabajadores ejecutan las pruebas de descifrado.

1. Lectura del archivo cifrado:
   El proceso maestro lee el archivo binario que contiene el texto cifrado (`ciphertext.bin`) y lo distribuye (broadcast) a todos los trabajadores. Esto

garantiza que todos los procesos tengan acceso al mismo bloque de datos cifrados.

2. Distribución dinámica del trabajo:
   El maestro no asigna de antemano todas las claves, sino que reparte *chunks* o bloques de claves conforme los trabajadores van solicitando trabajo.

3. Exploración pseudoaleatoria del espacio de claves:
   Dentro de cada bloque, el trabajador genera un recorrido pseudoaleatorio utilizando la función splitmix64() y un stride coprimo con el tamaño del bloque. Esto evita probar las claves en orden lineal y ayuda a distribuir la búsqueda de manera más equilibrada entre procesos.

4. Prueba de cada clave (`try_key`):
   Cada clave se convierte en un bloque DES de 8 bytes con paridad ajustada y se usa para descifrar el texto con **DES-ECB** (Electronic Codebook Mode).
   Luego, se busca si la palabra clave (`keyword`) aparece en el texto plano resultante. Si se encuentra coincidencia, el trabajador reporta la clave encontrada al maestro mediante `TAG_FOUND`.

5. Propagación de parada temprana:
   Cuando el maestro recibe una clave válida, envía un mensaje `TAG_STOP` a todos los procesos, indicando que deben detener la búsqueda inmediatamente.
   Esto evita seguir procesando claves innecesarias, reduciendo el tiempo total de ejecución.

6. Finalización:
   Si no se encuentra ninguna clave dentro del rango especificado, el maestro informa NOT FOUND in range.
   En caso de éxito, todos los procesos imprimen la clave encontrada (winner key) y finalizan ordenadamente.

DES Parallel Cracker

Un cracker paralelo por fuerza bruta para DES con MPI y OpenSSL. Lee un cipher (.bin), reparte el espacio de llaves entre procesos, prueba llaves en paralelo hasta encontrar una que, al descifrar, contenga la frase objetivo. Cuando algún proceso acierta, se propaga la parada, se mide el tiempo y el rank 0 imprime el resultado

Tecnicas de paralelizacion empleadas:

todos los procesos ejecutan el mismo programa en datos diferentes

Paralelismo de datos (embarassingly parallel)

Se reparte el espacio de llaves [0, 2^b) entre procesos MPI: cada rank prueba un subrango [mylower, myupper) de forma independiente (SPMD).

Partición estática del trabajo

División por bloques iguales (y ajuste en el último rank). Minimiza coordinación, maximiza tiempo en cómputo.