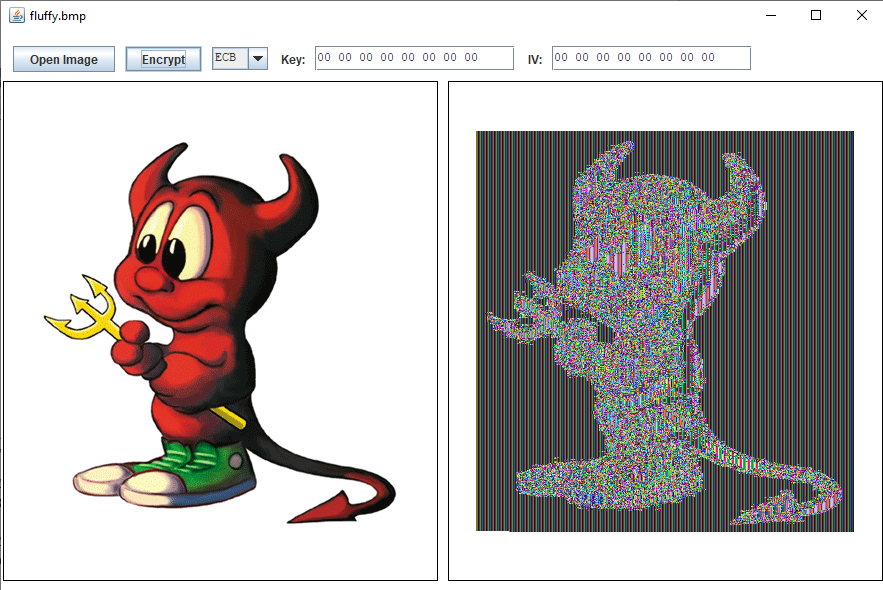
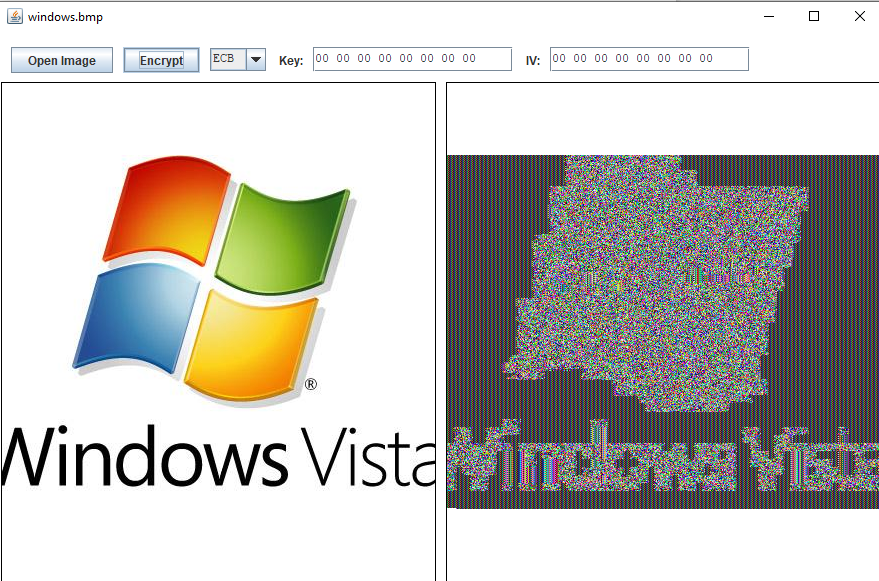
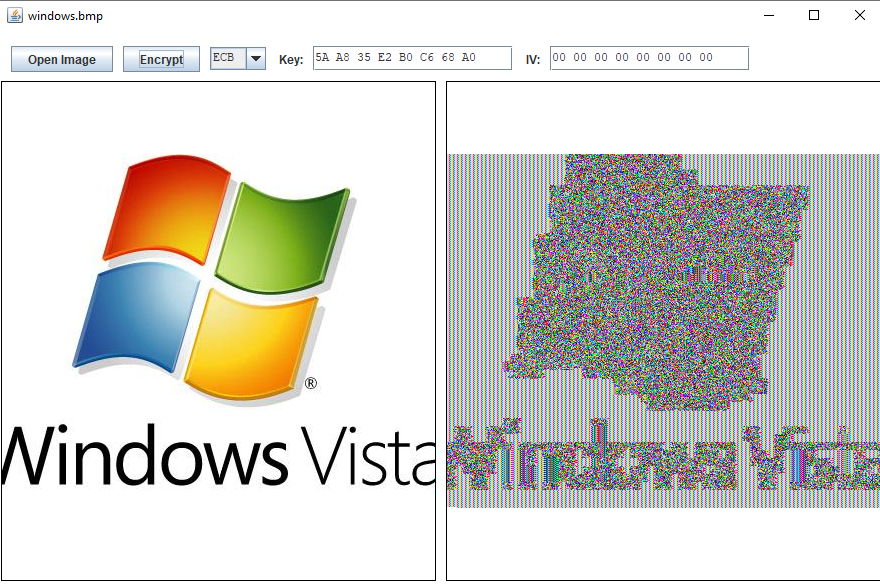
Taller “LaboratoryBlockCipherModes - Tux-Fluffy”.

Santiago Hernández Facio Lince 201922432

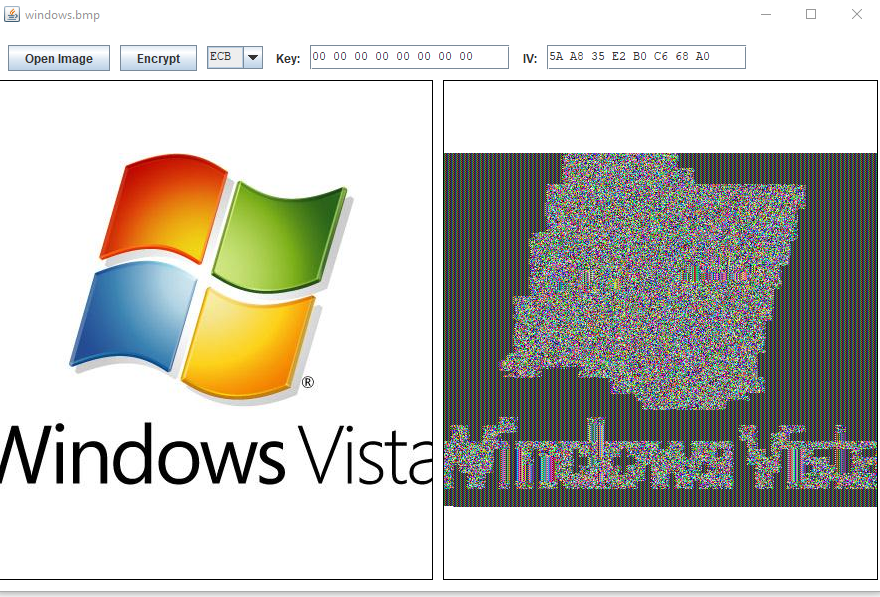
1. La imagen es cifrada con el modo de operación ECB con la llave: 00 00 00 00 00 00 00 00 y con el vector inicial: 00 00 00 00 00 00 00 00. Pese a que la imagen está cifrada, de todas formas, se puede notar la forma de la imagen original. Algo interesante de este modo de operación es que no importa la cantidad de veces que se cifre el mensaje inicial, si se mantiene la misma llave y el mismo vector inicial, el resultado siempre es el mismo:

1. Interfaz de usuario gráfica

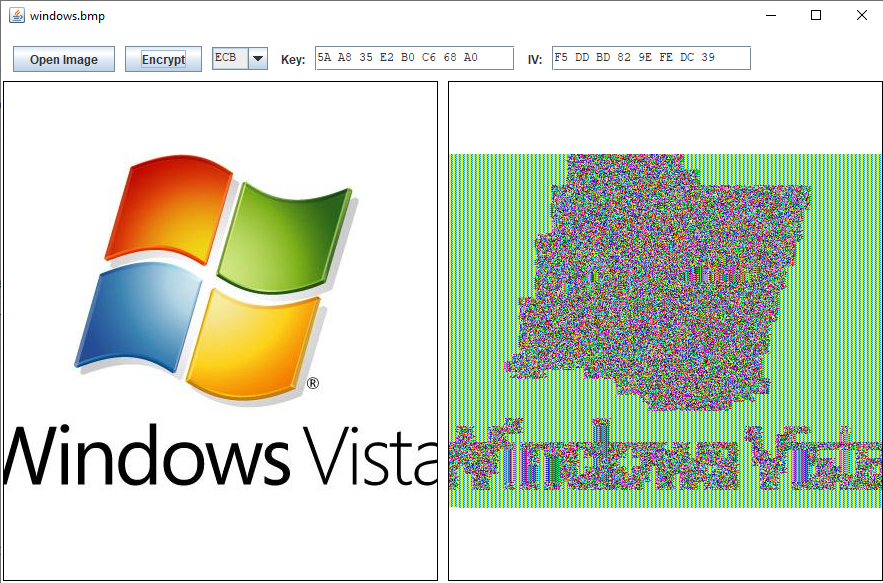
   Descripción generada automáticamenteCuando se cambia la llave dentro del modo ECB, el resultado se vuelve cada vez menos parecido con la imagen anterior, sin embargo, se sigue notando un patrón. En esta parte, usando un generador de números aleatorios hexadecimales, generé 2 llaves distintas: 5A A8 35 E2 B0 C6 68 A0, F5 DD BD 82 9E FE DC 39, este fue el resultado:

Como se puede ver en las imágenes, pese a que es una mejora sustancial con respecto a tener todo en 0 para la llave, se sigue notando un patrón bastante claro.

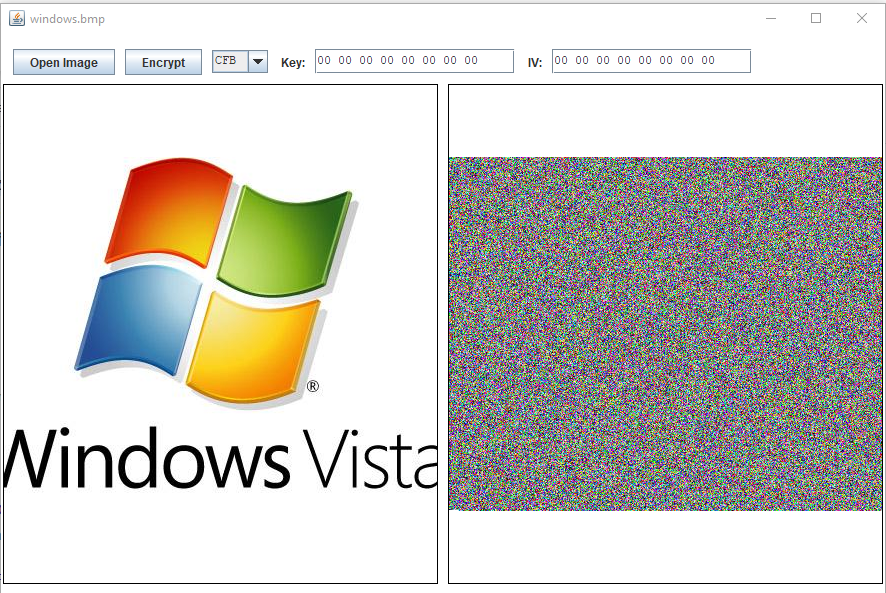
1. Para este apartado, empecé por regresar la contraseña a únicamente tener 0 y usé uno de los números aleatorios hexadecimales generados anteriormente como el vector inicial, el resultado sigue siendo un cifrado bastante inseguro ya que se sigue notando un patrón en la imagen:



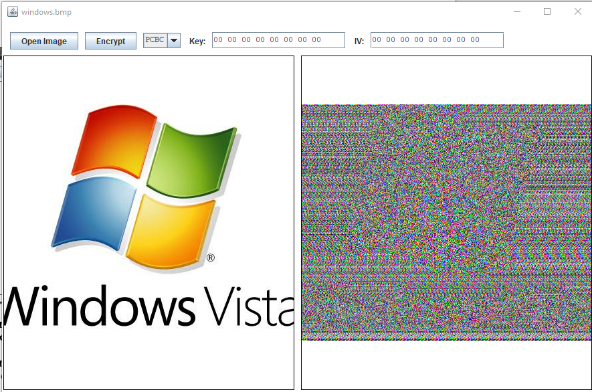
Posteriormente, haciendo todo tipo de combinaciones diferentes, con la llave y el vector inicial y por más aleatorio que fueran tanto la llave como el vector inicial, el resultado del cifrado seguía siendo insuficiente:



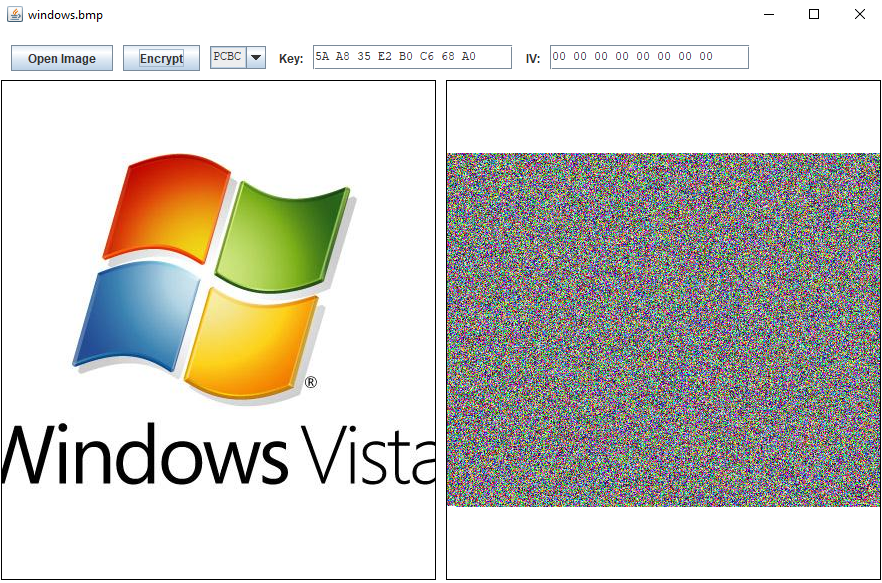
1. Al momento de usar los demás modos de operación (regresando la llave y el vector inicial a 0), fue muy interesante ver como CBC y CFB hace un trabajo mucho mejor de cifrado, borrando totalmente cualquier patrón de la imagen original y dando como resultado una imagen llena de pixeles sin sentido aparente:



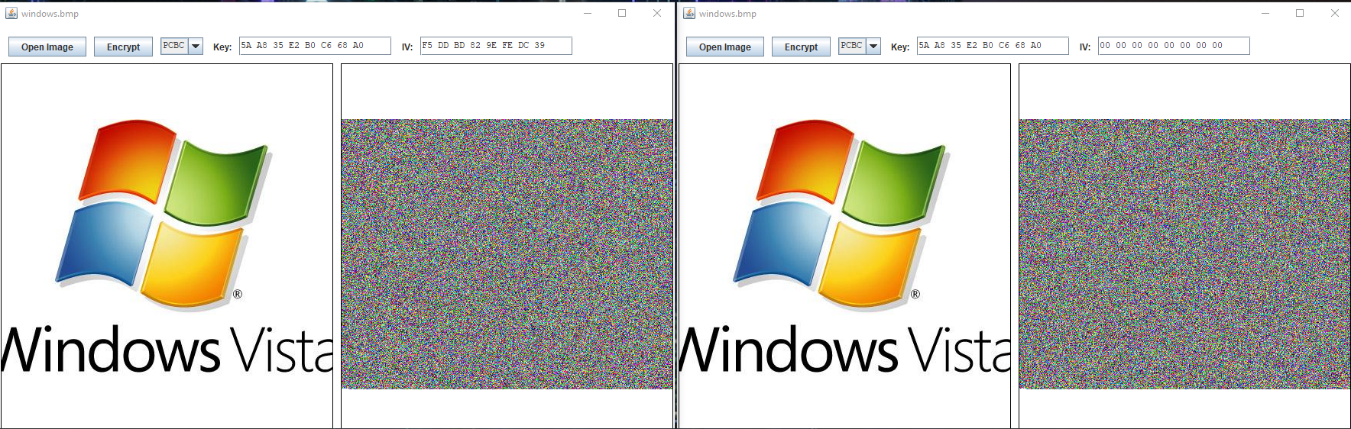
Por otro lado, al usar PCBC, se realiza un cifrado mucho mejor que con ECB, sin embargo, el resultado es peor que con CBC o CFB ya que aún se alcanza a notar un patrón mínimo de la imagen original:



1. Al cambiar la llave a: 5A A8 35 E2 B0 C6 68 A0, los modos CBC y CFB visualmente no se genera mayor cambio, se sigue obteniendo de resultado una imagen llena de pixeles que no presenta un patrón ni relación con la imagen original. Con el modo PCBC, que con la llave en 0 presentaba aún ciertos problemas de seguridad, al cambiar la llave, si genera un resultado seguro igual que los otros modos ya que la imagen obtenida, ya no presenta ningún tipo de patrón:

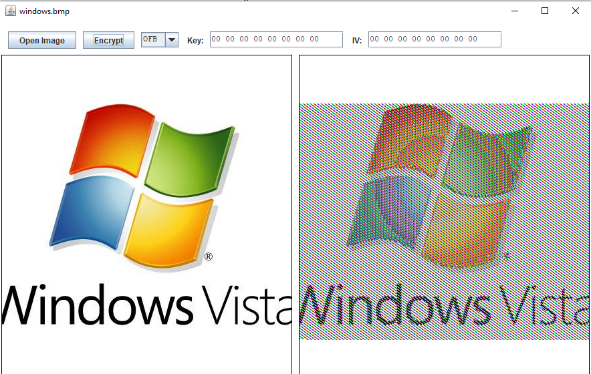


1. Añadiendo el siguiente vector de inicio: F5 DD BD 82 9E FE DC 39, los resultados a simple vista no se distinguen de cuando el vector de inicio era 0. Como anteriormente ya se tenía un nivel bueno de cifrado, al agregar este vector no se notan mayores cambios, acá una comparación visual del modo de operación PCBC (con vector inicial en 0 y con vector inicial en F5 DD BD 82 9E FE DC 39) que entre CBC, PCBC y CFB, como se vio anteriormente era el modo más débil de los 3:

Como se puede ver en la imagen, en ambos se obtiene un buen cifrado, sin ningún tipo de patrón en especial. Esto sucede de igual manera con los modos de operación CBC y CBF.

1. Cambiando al modo de operación a OFB y dejando el vector inicial y la llave en 0, obtenemos el peor resultado cifrado en comparación con los otros modos. Esto debido a que se puede notar casi completamente la imagen original. No es que se note algún patrón con respecto a la imagen inicial, sino que se nota casi que totalmente:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Esto se debe a que este modo de cifrado depende fuertemente en la llave y en el vector de inicialización, ya que estos juntos son cifrados y luego se realiza un XOR con el texto a cifrar. El problema recae en que, siendo que la llave y en el vector de inicialización tienen ambos un valor de 0, al momento de realizarse el cifrado entre los 2, no se genera una llave lo suficientemente “aleatoria” y al momento de realizar el XOR, genera un resultado similar a la imagen original.

El problema es que, como la llave para cifrar se genera a partir de la llave para cifrar anterior junto con la llave inicial y no junto al mensaje, como la llave inicial es 0, la llave para cifrar de cada bloque no presenta nunca un comportamiento aleatorio.

1. Al momento de cambiar el vector inicial a un valor que no sea 0, el resultado de la encriptación es considerablemente mejor, pero pese a que probé con varios vectores iniciales diferentes, generados de forma aleatoria, el número hexadecimal con el que tuve mejores resultados fue el siguiente: 89 11 A1 CF 43 3C 61 61, obteniendo el siguiente resultado:

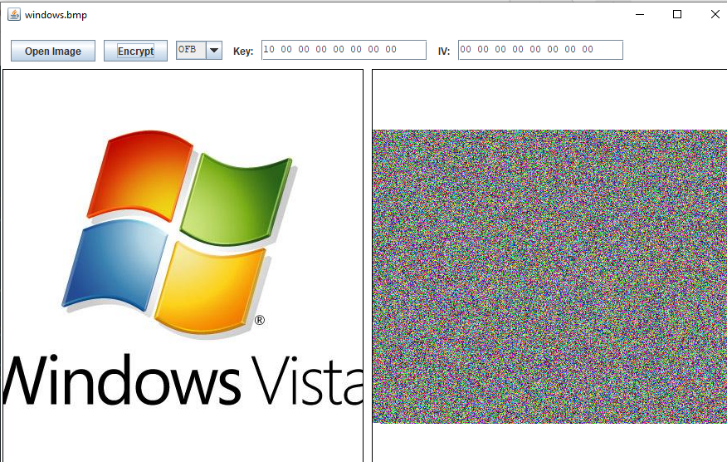
Interfaz de usuario gráfica

Descripción generada automáticamente

1. En este caso y para mi sorpresa, realizando un cambio mínimo en la llave inicial: 10 00 00 00 00 00 00 00 y dejando el vector inicial en 0, se logra obtener un resultado de cifrado igual de bueno a los anteriores. Esto se debe a que el vector inicial juega un papel importante ya que es la base para generar la primera llave para cifrar, pero de ahí en adelante tiene una influencia mínima, mientras que, por otro lado, la llave general, se usa en absolutamente cada iteración de OFB para generar la llave para cifrar. Con esto vemos que la llave general juega un rol enorme en la seguridad del modo, mostrando que una simple variación puede generar cambios gigantes en el resultado.

En este caso, cambiando un 0 por un 1, se genera la suficiente aleatoriedad para obtener un resultado cifrado satisfactorio.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

1. Estos modos son importantes en el cifrado de bloques ya que, por ejemplo, con DES, se debe usar un tamaño fijo de 64 bits tanto para el texto como para la llave, esto limita la cantidad de información que se puede cifrar, sin embargo, gracias a estos modos, se puede cifrar y descifrar la cantidad de información que se desee ya que definen todo el proceso sobre cómo manejar varios bloques de información para cifrarlos y descifrarlos con el uso de algún algoritmo como DES. Cada modo define este proceso de forma individual, presentando fortalezas y debilidades como mostrando en los anteriores numerales.
2. El mayor problema de ECB es que siempre que se use la misma llave, si se cifra X, el resultado siempre va a ser Z, generando siempre un resultado determinista, algo que no sucede con los otros modos.

Esto es particularmente inseguro ya que como hemos visto en las clases, el principal delator de los mensajes cifrados son los patrones que un criptoanalista enemigo puede aprovechar para hacer relaciones entre varios mensajes cifrados y eventualmente robar información confidencial o si tiene parejas de mensaje claro y cifrado puede incluso llegar a encontrar la llave. Por este problema, ECB no se recomienda para volúmenes grandes de datos, ya que se podrían encontrar muy seguidos estos patrones.

1. Realmente cifrar la cabecera resultaría en una mejora de seguridad ya que protegería la información que contiene de la imagen. Como estos modos de operación pueden manejar un número indefinido de bloques de datos, no tendrían problema con cifrar la cabecera también. Lo único que se debería hacer sería agregar padding a algún bloque para que todos queden del mismo tamaño. Esta mejora, no tendría mayor efecto en el tiempo de cifrado ni en el espacio en memoria que ocupe el resultado, esto debido a que las cabeceras de las imágenes tienen un tamaño en memoria pequeño.
2. Si, realizar un cifrado a imágenes JPEG o GIF es posible, pero hay que tener en cuenta que este tipo de archivos reducen la calidad de las imágenes para reducir la cantidad de memoria necesaria, por ende, al momento de descifrar el archivo, es muy posible que la imagen presente una calidad menor. Esto no tiene que ver con el algoritmo de cifrado/descifrado, es una particularidad de los archivos JPEG, GIF, etc.

Un tipo de imagen parecida a el 24-bit bitmap es el formato PNG, esto debido a que no pierde información al momento de realizar una compresión de la imagen a diferencia de lo que sucede con JPEG o GIF.

1. Esto se debe a que el cifrado DES se realiza por bloques de información de 64 bits mientras que en las imágenes cada píxel tiene 24 bits. por ende, en cada iteración que se use el algoritmo DES, se deben agrupar 64 bits para la encriptación, esto genera que se junten pixeles, perdiendo así el detalle de cada píxel y sobre todo, perdiendo precisión, generando los bloques que se ven en la imagen cifrada, especialmente en este caso ya que para llenar los 64 bits, toca tomar 2 pixeles y 16 bits del siguiente, dejando un pixel cifrado a medias, hasta la siguiente iteración.

Algo similar sucede cuando se quiere encontrar el área bajo la Gráfico, Histograma

Descripción generada automáticamentecurva de una función llenándola con bloques:

Ya que, aunque se logre encontrar una aproximación al valor del área, nunca será igual de preciso como si calculáramos la integral de la función en un rango definido.

1. Mientras más se ajuste el algoritmo de cifrado con la información a cifrar, el resultado será más preciso. Si DES fuera de 24 bits, se podría cifrar cada píxel de forma individual, generando un cifrado ajustado a la imagen original. Si DES tuviera un tamaño de 48 o 72, la cantidad de pixeles necesarios sería un número entero, en este caso 2 o 3, esto generaría una pérdida de precisión ya que por cada 2 o 3 píxeles, se generaría solo un bloque cifrado. En este caso, si la imagen original tiene 120 píxeles, con DES de 48 se generarían solo 60 bloques encriptados y con DES de 72 se tendrían 40. Esto no presenta mayor problema ya que la calidad de una imagen cifrada no es un factor importante, ya que la idea al cifrarla es que no se pueda reconocer lo que está en la imagen, por otro lado, reducir el tamaño del algoritmo DES si compromete la seguridad del resultado final.
2. Esto ocurre debido a que estos modos de operación generan dependencias entre los bloques, para cifrar el bloque x, es necesario tomar el bloque anterior x-1. Este funcionamiento tiene un problema y es que cuando se genera algún tipo de error, este es propagado en cada iteración mediante la operación XOR y considerando que los pixeles solo tienen 3 valores: RGB, al propagarse un error, los valores RGB pueden sufrir cambios, generalmente lo que sucede es que se baja la intensidad de alguno de los 3 valores y se llega a un valor de color “neutro”, en este caso el gris.

# Referencias

geeksforgeeks. (09 de 05 de 2023). *geeksforgeeks.org*. Obtenido de Block Cipher modes of Operation: https://www.geeksforgeeks.org/block-cipher-modes-of-operation/

IBM. (25 de 06 de 2021). *IBM*. Obtenido de Modes of operation: https://www.ibm.com/docs/en/zos/2.4.0?topic=data-modes-operation

Peter Loshin, M. C. (08 de 2021). *techtarget*. Obtenido de Data Encryption Standard (DES): https://www.techtarget.com/searchsecurity/definition/Data-Encryption-Standard#:~:text=The%20DES%20process%20involves%20encrypting,64%2Dbit%20key.

simplilearn. (09 de 02 de 2024). *www.simplilearn.com*. Obtenido de What Is DES (Data Encryption Standard)? DES Algorithm and Operation: https://www.simplilearn.com/what-is-des-article