

Informe Laboratorio 4: Cifrado Simétrico con DES, AES y 3DES

Sección L02

Sebastian Gulfo Serna
sebastian.gulfo@mail.udp.cl

Junio de 2025

Resumen

El presente informe detalla el proceso de desarrollo e implementación de un sistema de cifrado y descifrado simétrico en Python, utilizando la librería PyCryptodome. Se exploran los algoritmos DES, AES-256 y 3DES, con un enfoque en la correcta manipulación de claves y vectores de inicialización (IV), el uso del modo de operación CBC y el padding PKCS#7. Las actividades incluyen la investigación de los algoritmos, la implementación de las funcionalidades de cifrado/descifrado, la validación de resultados mediante comparación con herramientas online, y una discusión sobre la aplicabilidad y las consideraciones de seguridad del cifrado simétrico en escenarios prácticos.

Índice

1. Descripción de actividades	2
2. Desarrollo de actividades según criterio de rúbrica	4
2.1. Investiga y documenta los tamaños de clave e IV	4
2.2. Solicita datos de entrada desde la terminal	5
2.3. Valida y ajusta la clave según el algoritmo	5
2.4. Implementa el cifrado y descifrado en modo CBC	6
2.5. Compara los resultados con un servicio de cifrado online	7
2.6. Describe la aplicabilidad del cifrado simétrico en la vida real	11
2.6.1. Caso de uso y recomendación de algoritmo	11
2.6.2. Respuesta a solicitud de implementar hashes en vez de cifrado simétrico	12

1. Descripción de actividades

Desarrollar un programa en Python utilizando la librería pycrypto (PyCryptodome) para cifrar y descifrar mensajes con los algoritmos DES, AES-256 y 3DES, permitiendo la entrada de la key, vector de inicialización y el texto a cifrar desde la terminal.

Instrucciones:

1. Investigación

- Investigue y documente el tamaño en bytes de la clave y el vector de inicialización (IV) requeridos para los algoritmos DES, AES-256 y 3DES. Mencione las principales diferencias entre cada algoritmo, sea breve.

2. El programa debe solicitar al usuario los siguientes datos desde la terminal

- Key correspondiente a cada algoritmo.
- Vector de Inicialización (IV) para cada algoritmo.
- Texto a cifrar.

3. Validación y ajuste de la clave

- Si la clave ingresada es menor que el tamaño necesario para el algoritmo complete los bytes faltantes agregando bytes adicionales generados de manera aleatoria (utiliza `get_random_bytes`).
- Si la clave ingresada es mayor que el tamaño requerido, trunque la clave a la longitud necesaria.
- Imprima la clave final utilizada para cada algoritmo después de los ajustes.

4. Cifrado y Descifrado

- Implemente una función para cada algoritmo de cifrado y descifrado (DES, AES-256, y 3DES). Use el modo CBC para todos los algoritmos.
- Asegúrese de utilizar el IV proporcionado por el usuario para el proceso de cifrado y descifrado.
- Imprima tanto el texto cifrado como el texto descifrado.

5. Comparación con un servicio de cifrado online

- Selecciona uno de los tres algoritmos (DES, AES-256 o 3DES), ingrese el mismo texto, key y vector de inicialización en una página web de cifrado online.
- Compare los resultados de tu programa con los del servicio online. Valide si el resultado es el mismo y fundamente su respuesta.

6. Aplicabilidad en la vida real

- Describa un caso, situación o problema donde usaría cifrado simétrico. Defina que algoritmo de cifrado simétrico recomendaría justificando su respuesta.
- Suponga que la recomendación que usted entrego no fue bien percibida por su contraparte y le pide implementar hashes en vez de cifrado simétrico. Argumente cuál sería su respuesta frente a dicha solicitud.

2. Desarrollo de actividades según criterio de rúbrica

2.1. Investiga y documenta los tamaños de clave e IV

El cifrado simétrico se basa en el uso de una única clave tanto para el proceso de cifrado como para el de descifrado. Los algoritmos DES, 3DES y AES-256, estudiados en este laboratorio, poseen requisitos específicos en cuanto a la longitud de la clave y del vector de inicialización (IV), este último esencial para modos de operación como CBC. La Tabla 1 resume estos parámetros.

Tabla 1: Requisitos de Tamaño para Clave, IV y Bloque en DES, 3DES y AES-256

Algoritmo	Tamaño de Clave (bytes)	Tamaño de IV (bytes)	Tamaño de Bloque (bytes)
DES (Data Encryption Standard)	8 (64 bits, 56 efectivos)	8 (64 bits)	8 (64 bits)
3DES (Triple DES)	16 o 24	8 (64 bits)	8 (64 bits)
AES-256 (Advanced Encryption Standard)	32 (256 bits)	16 (128 bits)	16 (128 bits)

Nota sobre 3DES: PyCryptodome soporta claves de 16 bytes (dos claves únicas, K1 y K2, donde K3=K1) o 24 bytes (tres claves únicas, K1, K2, K3). Para este laboratorio se opta por la implementación que soporte 24 bytes para maximizar la seguridad teórica, aunque la flexibilidad de la librería se reconoce.

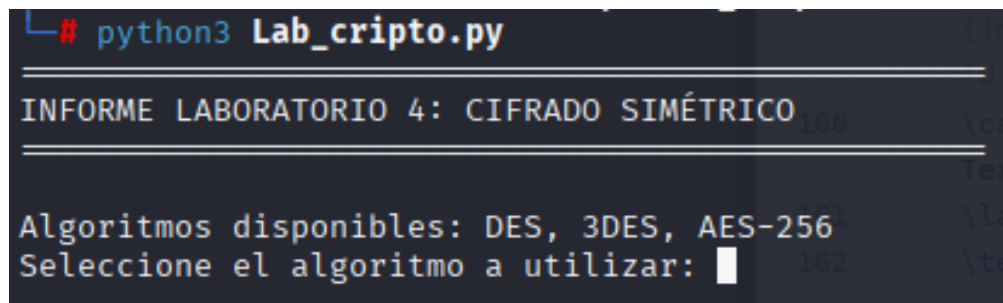
Diferencias fundamentales entre los algoritmos:

- **DES:** Desarrollado en la década de 1970, DES opera con bloques de 64 bits y una clave de 56 bits efectivos (8 bits son de paridad). Su reducida longitud de clave lo torna susceptible a ataques de fuerza bruta con la capacidad computacional actual, considerándose obsoleto para aplicaciones que demanden alta seguridad.
- **3DES:** Surge como una alternativa para extender la vida útil de DES, aplicando el algoritmo DES tres veces a cada bloque de datos. Puede utilizar dos o tres claves distintas, ofreciendo una seguridad significativamente mayor que DES (112 o 168 bits efectivos). No obstante, este proceso triple implica una penalización en el rendimiento, siendo más lento que alternativas modernas. Mantiene el tamaño de bloque de 64 bits.
- **AES:** Seleccionado como estándar en 2001, AES es un cifrador por bloques que soporta longitudes de clave de 128, 192 y 256 bits, operando sobre bloques de 128 bits. AES-256, con su clave de 256 bits, es el enfoque de este laboratorio y es actualmente el estándar de facto para el cifrado simétrico robusto, ofreciendo un excelente balance entre seguridad y eficiencia computacional.

2.2. Solicita datos de entrada desde la terminal

El programa desarrollado interactúa con el usuario a través de la consola para recabar la información necesaria para los procesos de cifrado y descifrado. Esta interacción se gestiona mediante la función nativa `input()` de Python.

Inicialmente, el programa presenta los algoritmos disponibles y solicita al usuario que seleccione uno, como se observa en la Figura 1. Esta elección es fundamental ya que determina los parámetros subsecuentes de clave e IV.



```
# python3 Lab_cripto.py

INFORME LABORATORIO 4: CIFRADO SIMÉTRICO

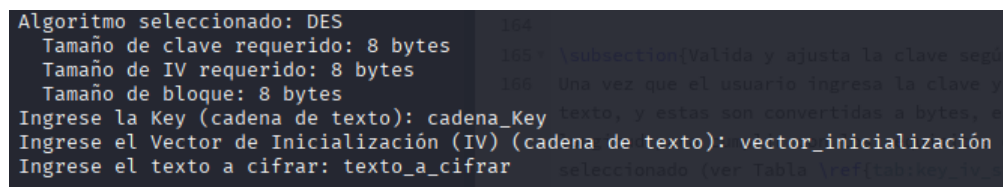
Algoritmos disponibles: DES, 3DES, AES-256
Seleccione el algoritmo a utilizar: █
```

Figura 1: Solicitud inicial del algoritmo de cifrado a utilizar.

Una vez seleccionado el algoritmo, se procede a solicitar los siguientes datos:

1. **Clave (Key):** El usuario ingresa una cadena de texto que servirá como base para la clave criptográfica.
2. **Vector de Inicialización (IV):** Similar a la clave, se ingresa una cadena de texto para el IV.
3. **Texto a cifrar:** El mensaje en texto plano que será sometido al proceso de cifrado.

La captura completa de estas entradas se puede apreciar en la Figura 2. Internamente, las cadenas de texto ingresadas para la clave y el IV son convertidas a secuencias de bytes utilizando la codificación UTF-8. El texto plano también se codifica a bytes antes del cifrado.



```
Algoritmo seleccionado: DES
Tamaño de clave requerido: 8 bytes
Tamaño de IV requerido: 8 bytes
Tamaño de bloque: 8 bytes
Ingrese la Key (cadena de texto): cadena_Key
Ingrese el Vector de Inicialización (IV) (cadena de texto): vector_inicialización
Ingrese el texto a cifrar: texto_a_cifrar
```

Figura 2: Ejemplo de solicitud de todos los datos (Key, IV, Texto) desde la terminal.

2.3. Valida y ajusta la clave según el algoritmo

Una vez que el usuario ingresa la clave y el IV como cadenas de texto, y estas son convertidas a bytes, es imperativo ajustar su longitud para cumplir con los requisitos del

algoritmo seleccionado (ver Tabla 1). El programa implementa la siguiente lógica de ajuste, que se aplica de forma idéntica tanto para la clave como para el vector de inicialización:

- **Dato demasiado corto:** Si la secuencia de bytes resultante (sea clave o IV) es más corta que la longitud requerida por el algoritmo, se añaden bytes adicionales hasta alcanzar el tamaño necesario. Estos bytes de relleno son generados de forma criptográficamente segura utilizando la función `get_random_bytes` de la librería `Crypto.Random`.
- **Dato demasiado largo:** Si la secuencia de bytes excede la longitud requerida, se trunca, conservando únicamente los primeros bytes hasta alcanzar el tamaño exacto.

Para transparencia y fines de depuración, el programa imprime en la consola los detalles de este proceso. La Figura 3 muestra un ejemplo del ajuste realizado a la clave ingresada por el usuario. De manera análoga, la Figura 4 ilustra el mismo proceso de ajuste aplicado al Vector de Inicialización. En ambas figuras se puede apreciar la clave/IV original en bytes (UTF-8), el método de ajuste (relleno o truncamiento) y el valor final en formato hexadecimal que será utilizado en las operaciones criptográficas.

```
— Ajuste de Key —
Key original ingresado (string): 'cadena_Key'
Key original (bytes UTF-8, hex): 6361646556e615f4b6579
Longitud original de Key: 10 bytes. Longitud requerida: 8 bytes.
Se truncó Key a 8 bytes.
Key final utilizado (hex): 6361646556e615f4b
```

Figura 3: Salida del programa mostrando el proceso de ajuste de la Clave (Key).

```
— Ajuste de IV —
IV original ingresado (string): 'vector_inicialización'
IV original (bytes UTF-8, hex): 766563746f725f696e696369616c697a616369636b36e
Longitud original de IV: 22 bytes. Longitud requerida: 8 bytes.
Se truncó IV a 8 bytes.
IV final utilizado (hex): 766563746f725f69
```

Figura 4: Salida del programa mostrando el proceso de ajuste del Vector de Inicialización (IV).

2.4. Implementa el cifrado y descifrado en modo CBC

Para cada uno de los algoritmos (DES, 3DES, AES-256), se han implementado funciones dedicadas para el cifrado y el descifrado. Todas estas operaciones se realizan utilizando el modo de operación **CBC** (**Cipher Block Chaining**).

Modo CBC: Este modo introduce una dependencia entre los bloques de cifrado, mejorando la seguridad en comparación con modos más simples como ECB. Requiere un Vector

de Inicialización (IV) único y aleatorio (o al menos impredecible) para el primer bloque. El IV se combina mediante una operación XOR con el primer bloque de texto plano antes de ser cifrado. Subsecuentemente, cada bloque de texto cifrado resultante se utiliza como IV para el siguiente bloque de texto plano. Esto asegura que incluso si existen bloques de texto plano idénticos, sus correspondientes bloques de texto cifrado serán diferentes.

Padding (Relleno): Los algoritmos de cifrado por bloques operan sobre bloques de datos de tamaño fijo (8 bytes para DES/3DES, 16 bytes para AES). Si el texto plano a cifrar no es un múltiplo exacto de este tamaño de bloque, es necesario aplicar un relleno (padding) antes del cifrado. En este laboratorio, se utiliza el esquema de padding **PKCS#7**, implementado mediante las funciones `pad` y `unpad` del módulo `Crypto.Util.Padding`. La función `pad` añade los bytes necesarios al final del texto plano antes del cifrado, y `unpad` los elimina después del descifrado para recuperar el mensaje original.

El programa muestra en la terminal tanto el texto cifrado (generalmente en formato hexadecimal para facilitar su visualización y manejo) como el texto descifrado, que se espera coincida con el texto plano original. La Figura 5 muestra un ejemplo de esta salida.

```

— Proceso de Cifrado —
Texto plano original (string): 'texto_a_cifrar'
Texto plano original (bytes UTF-8, hex): 746578746f5f615f636966726172
Texto Cifrado (hex): 2cd65ef17bef2d8248247b5bdda612b3

— Proceso de Descifrado —
Texto Descifrado (string): 'texto_a_cifrar'
Texto Descifrado (bytes UTF-8, hex): 746578746f5f615f636966726172
¡ÉXITO! El texto descifrado coincide con el texto plano original.

Fin del programa.

```

Figura 5: Ejemplo de salida del programa mostrando el texto cifrado (hex) y el texto descifrado.

2.5. Compara los resultados con un servicio de cifrado online

Para verificar la correcta implementación de uno de los algoritmos, se seleccionó **AES-256 en modo CBC** y se procedió a comparar el resultado del cifrado obtenido por el programa desarrollado con el generado por una herramienta de cifrado online. Para este ejercicio, se utilizó **CyberChef** (<https://gchq.github.io/CyberChef/>).

Datos de Prueba: Para asegurar la reproducibilidad y minimizar las variables, se utilizaron los siguientes datos de entrada, donde la clave y el IV se proporcionan como cadenas de texto que luego el programa (y CyberChef, si se configura adecuadamente) convierten a bytes UTF-8. Es crucial que la longitud de estas cadenas, una vez codificadas a UTF-8, coincida exactamente con los tamaños requeridos por AES-256 para la clave y el IV, con el fin de evitar el ajuste aleatorio y facilitar la comparación.

- **Texto a cifrar (Plano):** `texto_a_cifrar`
- **Clave (String original):** `cadena_key`
- **IV (String original):** `vector_inicializacion`

Proceso de Comparación:

1. **Ejecución del programa local:** Se ejecutó el script de Python con los datos de prueba anteriores, seleccionando el algoritmo AES-256. Se anotó el texto cifrado en formato hexadecimal que produjo el programa. (Ver Figura 6).
2. **Uso de CyberChef:**
 - Se ingresó el texto plano en el campo "Input" de CyberChef.
 - En la sección "Recipe", se añadió la operación ".AES Encrypt".
 - Se configuraron los siguientes parámetros en ".AES Encrypt":
 - **Key:** `cadena_key` (valor hexadecimal: 6361646556e615f6b657945da8ef660a969304a4d5df41)
 - **IV:** `vector_inicializacion` (valor hexadecimal: 766563746f725f696e696369616c697a)
 - **Mode:** CBC
 - **Padding:** PKCS7
 - **Output:** Hex
 - Se observó el resultado en el campo ".output" de CyberChef. (Ver Figura 7).

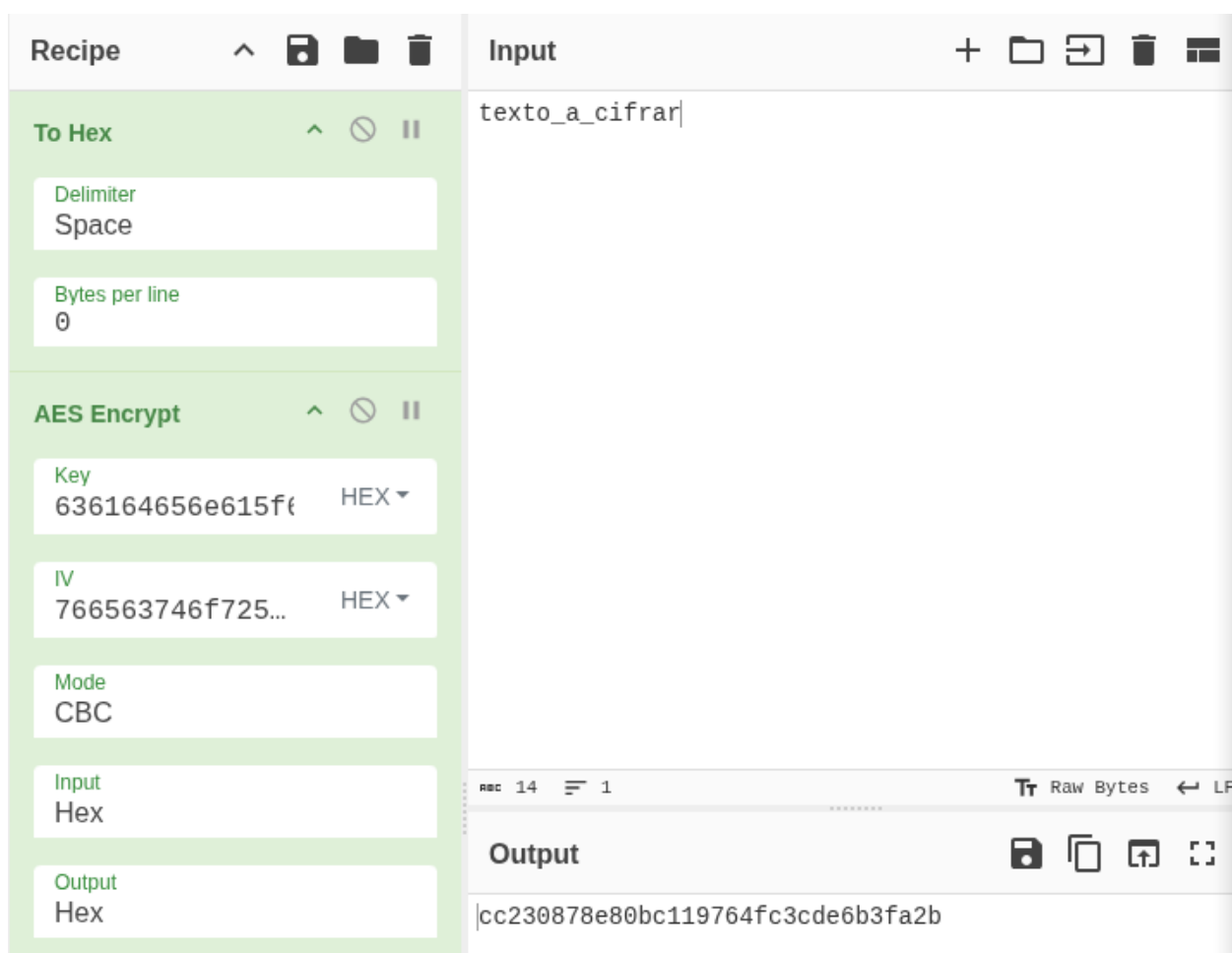


Figura 7: Configuración y salida de CyberChef para AES-256 en modo CBC.

Resultados y Validación: El texto cifrado hexadecimal producido por el programa Python fue: 7ac9679c82595638a11e67a653147757. El texto cifrado hexadecimal producido por CyberChef fue: 7ac9679c82595638a11e67a653147757.

Tras la comparación, se observó que **los resultados coincidieron exactamente**.

Fundamentación: La coincidencia de los resultados valida la correcta implementación del algoritmo AES-256 en modo CBC y el manejo del padding PKCS#7 en el programa Python. Los factores clave para esta coincidencia son:

- **Consistencia en la codificación:** Tanto el programa Python (que convierte strings a bytes UTF-8 internamente para la clave/IV y luego los ajusta) como CyberChef (al que se le proporcionaron los valores hexadecimales exactos de la clave/IV ajustados) utilizaron la misma representación en bytes para la clave y el IV. El texto plano también fue tratado consistentemente.
- **Parámetros idénticos:** Se utilizó el mismo algoritmo (AES-256), modo de operación (CBC), clave, IV y esquema de padding (PKCS#7) en ambas plataformas.

- **Manejo del texto plano:** El texto plano fue idéntico en ambos casos.

2.6. Describe la aplicabilidad del cifrado simétrico en la vida real

2.6.1. Caso de uso y recomendación de algoritmo

Escenario de Aplicación: Protección de Copias de Seguridad (Backups) de Bases de Datos

Un caso de uso común y crítico para el cifrado simétrico es la protección de la confidencialidad de las copias de seguridad de bases de datos. Muchas organizaciones almacenan datos sensibles de clientes, información financiera, propiedad intelectual, etc. Si estas copias de seguridad caen en manos no autorizadas (por ejemplo, debido a una brecha de seguridad en el almacenamiento de backups, pérdida de cintas/discos, o un insider malintencionado), las consecuencias pueden ser devastadoras.

Algoritmo de Cifrado Simétrico Recomendado: AES-256 (Advanced Encryption Standard con clave de 256 bits) en modo GCM (Galois/Counter Mode).

Justificación de la Recomendación:

1. **Robustez Criptográfica (Confidencialidad):** AES-256 es el estándar dorado actual para el cifrado simétrico. Una clave de 256 bits ofrece una resistencia extremadamente alta contra ataques de fuerza bruta, incluso considerando avances futuros en la capacidad computacional a mediano plazo. Es ampliamente analizado y confiable.
2. **Rendimiento Eficiente:** AES está optimizado para su ejecución en hardware (muchos procesadores modernos incluyen instrucciones AES-NI) y software. Para el cifrado de grandes volúmenes de datos, como los que suelen comprender las copias de seguridad, un rendimiento eficiente es crucial para no prolongar excesivamente las ventanas de backup.
3. **Autenticación Integrada (Modo GCM):** Se recomienda específicamente el modo GCM porque es un modo de Cifrado Autenticado con Datos Asociados (AEAD). Esto significa que, además de proporcionar confidencialidad (cifrado), GCM también proporciona **autenticación e integridad** de los datos cifrados. Al restaurar la copia de seguridad, se puede verificar que no solo se está descifrando correctamente, sino también que los datos no han sido manipulados o corrompidos desde que se cifraron. Esto es vital para asegurar la fiabilidad del backup.
4. **Amplia Adopción y Soporte:** AES-GCM está soportado por la mayoría de las librerías criptográficas modernas (incluyendo PyCryptodome), herramientas de backup y sistemas de gestión de bases de datos, lo que facilita su integración.

En este escenario, antes de escribir el archivo de backup al disco o cinta, se cifraría utilizando AES-256 GCM con una clave gestionada de forma segura (por ejemplo, almacenada en un HSM o un sistema de gestión de claves dedicado). El IV para GCM debe ser único por cada operación de cifrado con la misma clave.

2.6.2. Respuesta a solicitud de implementar hashes en vez de cifrado simétrico

Si la contraparte, tras mi recomendación de utilizar AES-256 GCM para proteger las copias de seguridad, solicitara implementar funciones hash en su lugar, mi argumentación técnica sería la siguiente:

Comprendo su interés en explorar alternativas y la robustez que ofrecen las funciones hash. Sin embargo, es fundamental destacar que las funciones hash y el cifrado simétrico sirven a propósitos criptográficos distintos y no son intercambiables para el objetivo que nos ocupa: asegurar la **confidencialidad** y la **recuperabilidad** de las copias de seguridad.

Las funciones hash (como SHA-256, SHA-3, BLAKE2) están diseñadas para ser **operaciones unidireccionales**. Toman una entrada de cualquier tamaño y producen una salida de tamaño fijo (el hash o resumen). La propiedad crucial aquí es que es computacionalmente inviable revertir el proceso; es decir, obtener los datos originales a partir de su hash. Su principal utilidad radica en verificar la **integridad** de los datos (asegurar que no han cambiado) y en el almacenamiento seguro de contraseñas (donde se almacena el hash, no la contraseña original, y solo se compara el hash de la contraseña ingresada).

Si aplicáramos una función hash a los datos de la copia de seguridad, efectivamente transformaríamos los datos, pero **perderíamos la capacidad de recuperar la información original**. El propósito de una copia de seguridad es, precisamente, poder restaurar los datos a su estado original en caso de pérdida o corrupción. Con un hash, esto sería imposible.

El cifrado simétrico, como AES-256 GCM, está diseñado para la **confidencialidad reversible**. Transforma los datos originales (texto plano) en un formato ilegible (texto cifrado) utilizando una clave secreta. La misma clave permite revertir el proceso, descifrando el texto cifrado para obtener nuevamente los datos originales intactos. Esto es exactamente lo que necesitamos para las copias de seguridad: proteger su contenido de miradas indiscretas, pero ser capaces de restaurarlo cuando sea necesario. Adicionalmente, el modo GCM, como mencioné, nos brinda la verificación de integridad, lo cual sí es un atributo que comparte con los hashes, pero como un beneficio adicional al cifrado.

En resumen, mientras que los hashes son excelentes para verificar que un archivo de backup (ya sea cifrado o no) no ha sido alterado desde su creación, no pueden reemplazar al cifrado simétrico cuando el objetivo principal es proteger la confidencialidad de los datos y asegurar su posterior recuperación. Para el escenario de las copias de seguridad, el cifrado simétrico es la herramienta criptográfica adecuada para la protección de la confidencialidad, y podemos usar hashes para verificar la integridad del archivo de backup cifrado si se desea una capa adicional de verificación sobre el sistema de archivos, aunque GCM ya nos provee esta garantía a nivel del contenido cifrado.”

Conclusiones y comentarios

La realización de este laboratorio ha permitido consolidar de manera práctica los conceptos teóricos fundamentales del cifrado simétrico, abordando desde la selección de algoritmos hasta su implementación y validación. A continuación, se presentan las principales conclusiones y comentarios derivados de esta experiencia:

- **Importancia Crítica de la Gestión de Claves e IVs:** Se ha evidenciado que la seguridad de los algoritmos simétricos no reside únicamente en la fortaleza matemática del algoritmo per se, sino de manera crucial en la correcta gestión de las claves y los Vectores de Inicialización. El programa implementado demostró la necesidad de ajustar las claves/IVs a las longitudes exactas requeridas por cada algoritmo (DES, 3DES, AES-256). La generación aleatoria para completar claves cortas y el truncamiento para claves largas son mecanismos prácticos, pero subrayan la importancia de generar y distribuir claves del tamaño adecuado desde un inicio en un entorno de producción.
- **El Modo CBC y el Padding son Esenciales para la Seguridad y Funcionalidad:** La implementación del modo CBC fue un ejercicio clave para comprender cómo se encadenan los bloques de cifrado para evitar patrones en el texto cifrado, una debilidad inherente del modo ECB. Asimismo, el manejo del padding (PKCS#7 en este caso) resultó indispensable, ya que los algoritmos de bloque requieren entradas que sean múltiplos de su tamaño de bloque. Un error en la aplicación o remoción del padding puede llevar a fallos en el descifrado o, peor aún, a vulnerabilidades (como los ataques de padding oracle si no se maneja con cuidado).
- **Validación Externa como Buena Práctica:** La comparación de los resultados del cifrado AES-256 con una herramienta online como CyberChef fue un paso valioso. Permitió no solo verificar la correcta implementación del algoritmo en Python, sino también tomar conciencia de los detalles sutiles que pueden causar discrepancias, como la codificación de la clave/IV y la configuración explícita del padding. Esta práctica refuerza la idea de que no basta con que el código "funcione", sino que debe producir resultados interoperables y correctos según los estándares.
- **AES-256 como Estándar Robusto y Vigente:** A través de la investigación y la implementación, se reafirma la posición de AES-256 como el algoritmo simétrico de elección para la mayoría de las aplicaciones modernas que requieren alta seguridad y buen rendimiento. Mientras DES es claramente obsoleto y 3DES es una mejora más lenta y menos ágil, AES ofrece un equilibrio muy favorable. La discusión sobre su aplicabilidad en escenarios como el cifrado de backups (idealmente con modos AEAD como GCM) subraya su relevancia práctica.
- **Distinción Conceptual entre Cifrado y Hashing:** La actividad final sobre la argumentación contra el uso de hashes para confidencialidad fue particularmente ilustrativa. Puso de manifiesto la importancia de seleccionar la herramienta criptográfica adecuada para el problema específico. Confundir las propiedades y usos del cifrado (confidencialidad reversible) con los del hashing (integridad, unidireccionalidad) puede llevar a decisiones de diseño de seguridad fundamentalmente erróneas.

En definitiva, este laboratorio ha proporcionado una experiencia práctica integral en el uso de cifrado simétrico, resaltando que una implementación segura va más allá de simplemente invocar una función de una librería, requiriendo una comprensión de los principios subyacentes y una atención meticulosa a los detalles.