Informe Laboratorio 4

Sección 2

Ignacio Pastén
e-mail: ignacio.pasten@mail.udp.cl

Octubre de 2025

Índice

1. Descripción de actividades

Desarrollar un programa en Python utilizando la librería pycrypto 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

Para los algoritmos de cifrado simétrico utilizados, los tamaños de clave (key) e inicializador de vector (IV) son los siguientes:

- DES (Data Encryption Standard): clave de 8 bytes (64 bits), de los cuales 56 son efectivos y 8 corresponden a bits de paridad. El tamaño de bloque y del IV es de 8 bytes.
- 3DES (Triple DES): clave de 24 bytes (192 bits), formada por tres claves DES (64 bits) aplicadas secuencialmente (EDE: encriptar-desencriptar-encriptar). El bloque y el IV son de 8 bytes.
- AES-256 (Advanced Encryption Standard): clave de 32 bytes (256 bits), con bloque e IV de 16 bytes. Ofrece mayor seguridad y velocidad en comparación con DES y 3DES.

La principal diferencia entre DES, AES-256 y 3DES-reside en su evolución, longitud de clave y rendimiento. DES es el algoritmo más antiguo y se considera obsoleto debido a su clave corta (56 bits), lo que compromete su seguridad. 3DES surgió como una mejora al aplicar DES tres veces, aumentando la seguridad a costa de la eficiencia computacional. Por último, AES-256 representa el estándar moderno, superando a ambos en seguridad (con una clave de 256 bits) y en velocidad, consolidándose como la opción preferida y más eficiente hoy en día.

2.2. Solicita datos de entrada desde la terminal

Para el desarrollo de la actividad se utizará un programa hecho con Python, dicho c ´digo solicita los datos al usuario directamente desde la terminal. Los valores ingresados pueden variar según el algoritmo seleccionado, y el sistema está diseñado para responder correctamente ante distintos casos de entrada.

lab_4.py U

lab_4.py > ...

```python
#!/usr/bin/env python3
import base64
from typing import Tuple
try:
    from Cryptodome.Cipher import DES, DES3, AES
    from Cryptodome.Random import get_random_bytes
    from Cryptodome.Util.Padding import pad, unpad
except Exception as e:
    print("ERROR")
    raise

BLOCK_SIZES = {
    "DES": 8,
    "3DES": 8,
    "AES-256": 16,
}

KEY_SIZES = {
    "DES": 8,          # 64 bits
    "3DES": 24,        # 192 bits
    "AES-256": 32,     # 256 bits
}

def ensure_bytes(s: str) -> bytes:

    s = s.strip()
    if s.lower().startswith("hex:"):
        return bytes.fromhex(s[4:])
    if s.lower().startswith("0x"):
        return bytes.fromhex(s[2:])
    return s.encode("utf-8")

def adjust_key(alg: str, key: bytes) -> bytes:
    target = KEY_SIZES[alg]
    if alg == "3DES":

        if len(key) < 24:
            from Cryptodome.Random import get_random_bytes
            key = key + get_random_bytes(24 - len(key))
        elif len(key) > 24:
            key = key[:24]
```

Figura 1: Código Python.

El programa cifra y descifra mensajes con DES, 3DES y AES-256 en modo CBC, ajustando la clave y el IV automáticamente. Muestra el texto cifrado y descifrado, comprobando que el proceso sea correcto y reversible.

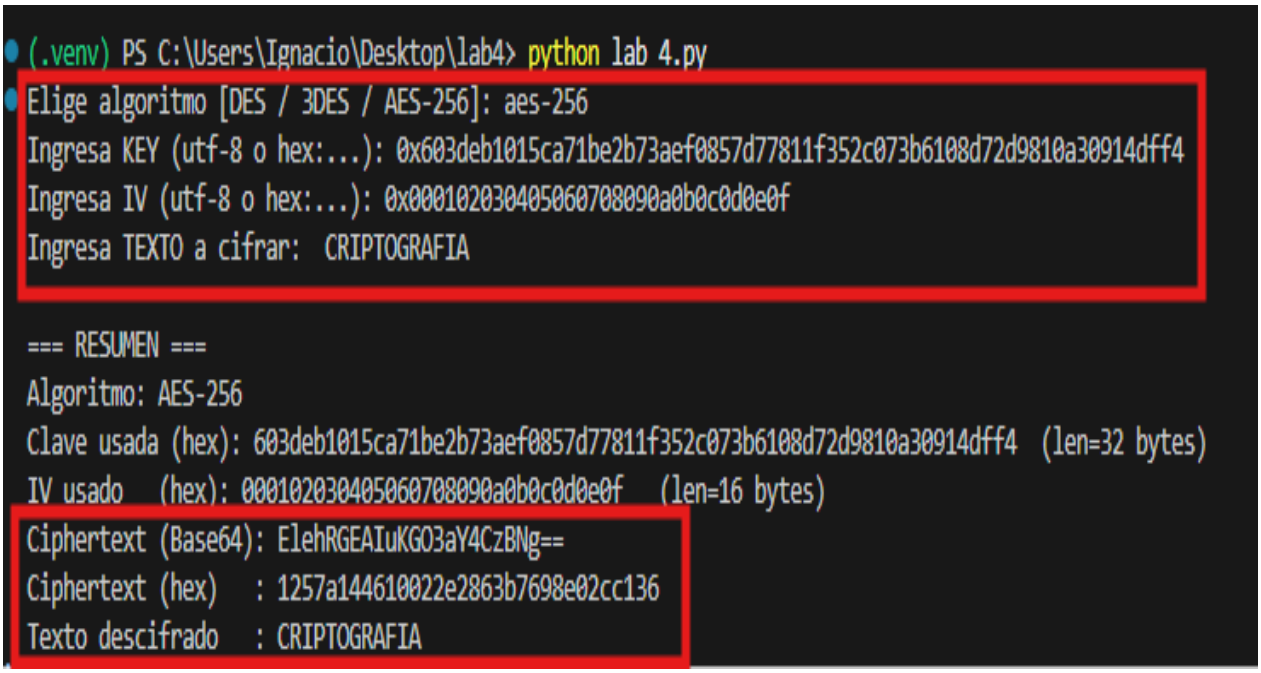

Figura 2: Programa solicitando datos desde la terminal e imprimiendo los resultados.

Es importante que los valores se introduzcan en el formato correcto. Si la entrada está en formato hexadecimal, debe anteponerse el prefijo `0x`, de lo contrario, el programa asumira que el dato se encuentra en formato de texto (*UTF-8*). Esta validación evita resultados erroneos en la interpretación de la clave o del vector de inicialización, garantizando la correcta ejercución del cifrado. Esto permite realizar las pruebas que se desee para cada algoritmo.

2.3. Valida y ajusta la clave según el algoritmo

El programa implementa una función llamada `adjust_key()`, encargada de validar y ajustar la longitud de la clave según los requerimientos del algoritmo seleccionado. Si la clave es menor, se completa automáticamente con bytes aleatorios generados por la función `get_random_bytes()`; si es mayor, se trunca al tamaño necesario. Finalmente, se imprime la clave ajustada en formato hexadecimal, junto con su longitud.

A continuación se muestran dos ejemplos representativos del proceso:

- **Caso 1 – Clave más corta de lo requerido (AES-256):** se completaron los bytes faltantes hasta los 32 bytes exigidos.

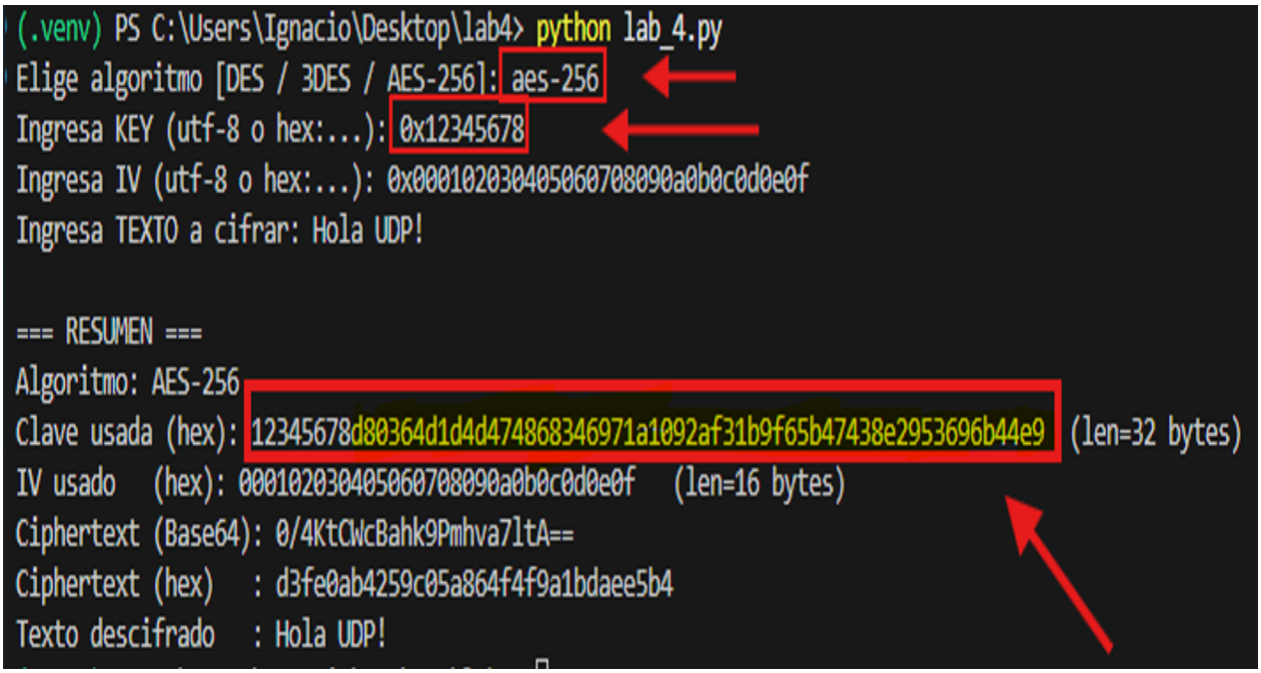

Figura 3: Ajuste de clave corta para AES-256 mediante la función `get_random_bytes()`.

Cabe mencionar que, si se vuelve a ejecutar con la misma entrada, la clave final será distinta (porque los bytes aleatorios cambian). Esto demuestra claramente el uso de la función `get_random_bytes()`.

- **Caso 2 – Clave más larga de lo requerido (3DES):** se truncó la clave a 24 bytes exactos.

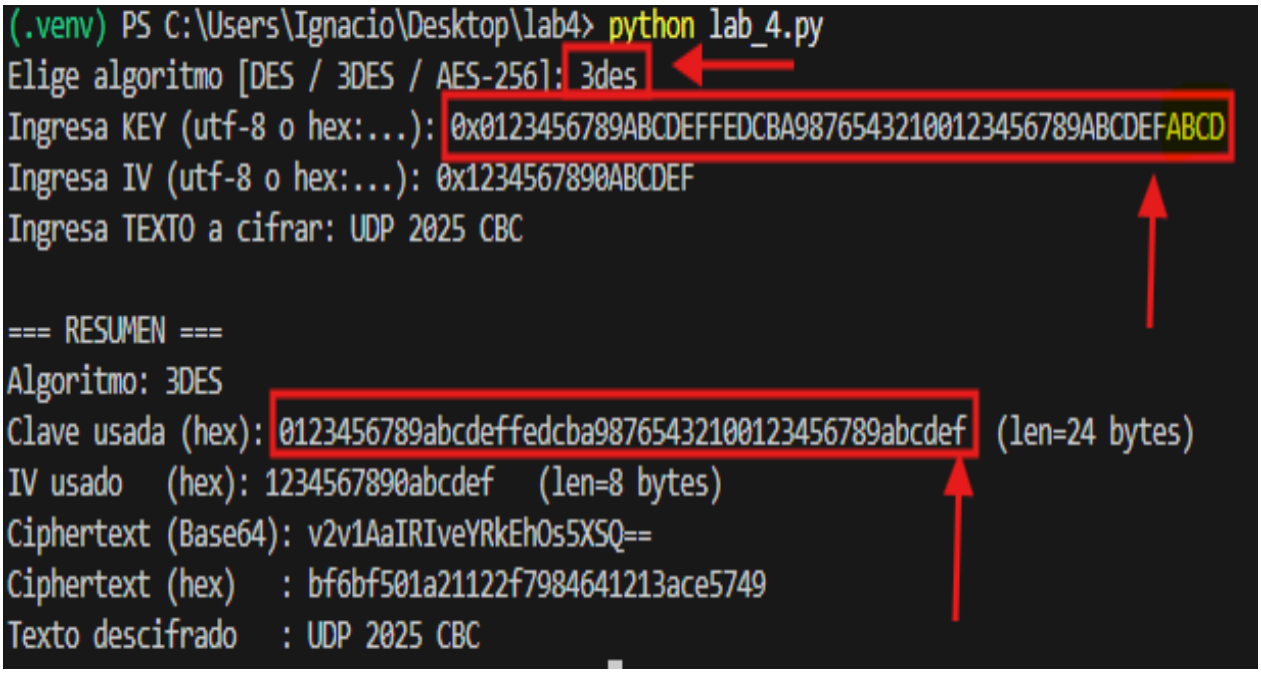

Figura 4: Truncamiento de una clave larga para 3DES hasta los 24 bytes requeridos.

En esta ejecución, la clave original tenía más de 24 bytes y el programa muestra solo los primeros 24 bytes usados, demostrando que el truncamiento se realizó correctamente.

A continuación, la siguiente tabla resume los tamaños de clave necesarios y los comportamientos de ajuste implementados por el programa:

Algoritmo	**Tamaño requerido**	**Clave corta**	**Clave larga**
DES	8 bytes	Completa con aleatorios	Trunca a 8 bytes
3DES	24 bytes	Completa con aleatorios y ajusta paridad	Trunca a 24 bytes
AES-256	32 bytes	Completa con aleatorios	Trunca a 32 bytes

Tabla 1: Comportamiento de validación y ajuste de clave para cada algoritmo.

Con este proceso se garantiza que las claves utilizadas sean compatibles con los estándares de cada algoritmo y que el cifrado se ejecute correctamente sin errores.

2.4. Implementa el cifrado y descifrado en modo CBC

El programa implementa funciones específicas para realizar el cifrado y el descifrado de mensajes utilizando el modo **CBC (Cipher Block Chaining)**. Este modo de operación encadena los bloques de texto cifrado, garantizando que una pequeña modificación en el texto original o en el vector de inicialización (IV) altere significativamente el resultado final, aumentando así la seguridad.

El proceso se desarrolla en dos funciones principales:

- **Cifrado:** utiliza la función `pad()` para aplicar el relleno (*padding*) PKCS7, asegurando que el texto tenga una longitud múltiplo del tamaño de bloque. Luego, con la función `cipher.encrypt()`, se obtiene el texto cifrado.

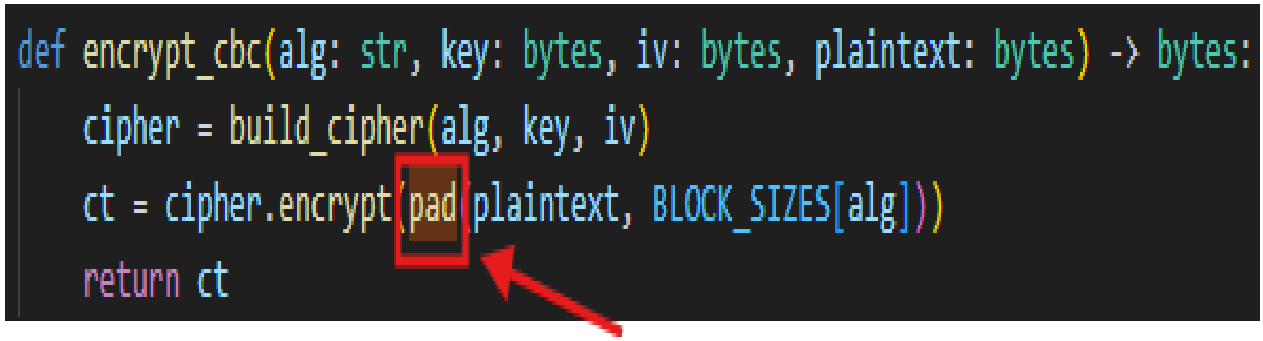

Figura 5: Función cifrado.

- **Descifrado:** aplica `cipher.decrypt()` y posteriormente la función `unpad()` para eliminar el relleno y recuperar el texto original sin alteraciones.

Figura 6: Función decifrado.

A continuación se presentan los resultados obtenidos al ejecutar el programa con los tres algoritmos implementados: AES-256, 3DES y DES, todos en modo CBC.

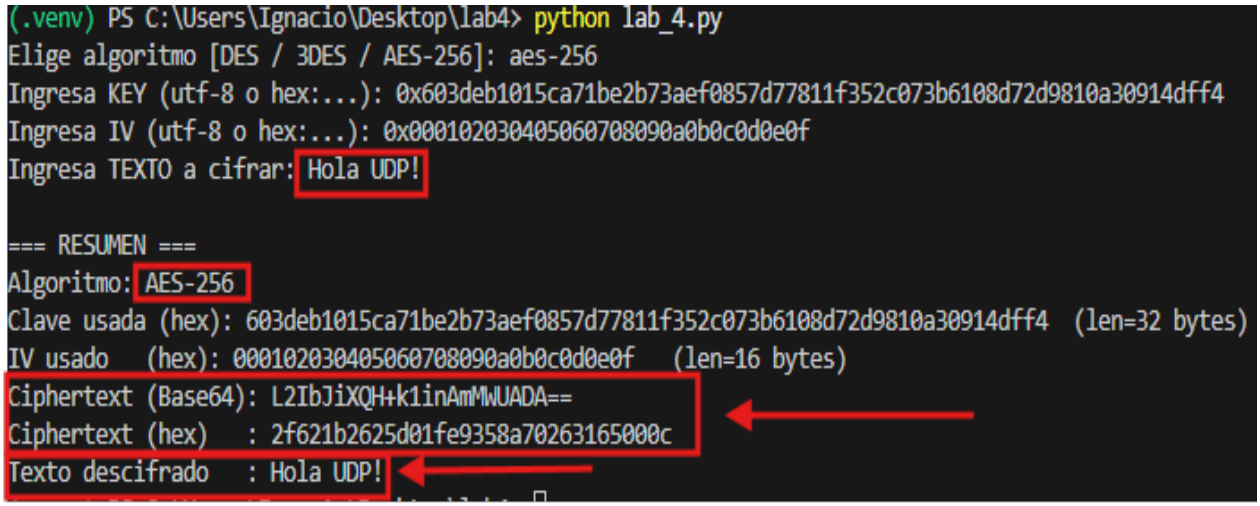

Figura 7: Cifrado y descifrado con el algoritmo AES-256 en modo CBC

Se observa que el texto descifrado coincide exactamente con el texto original, verificando la correcta implementación del cifrado y descifrado en modo CBC.

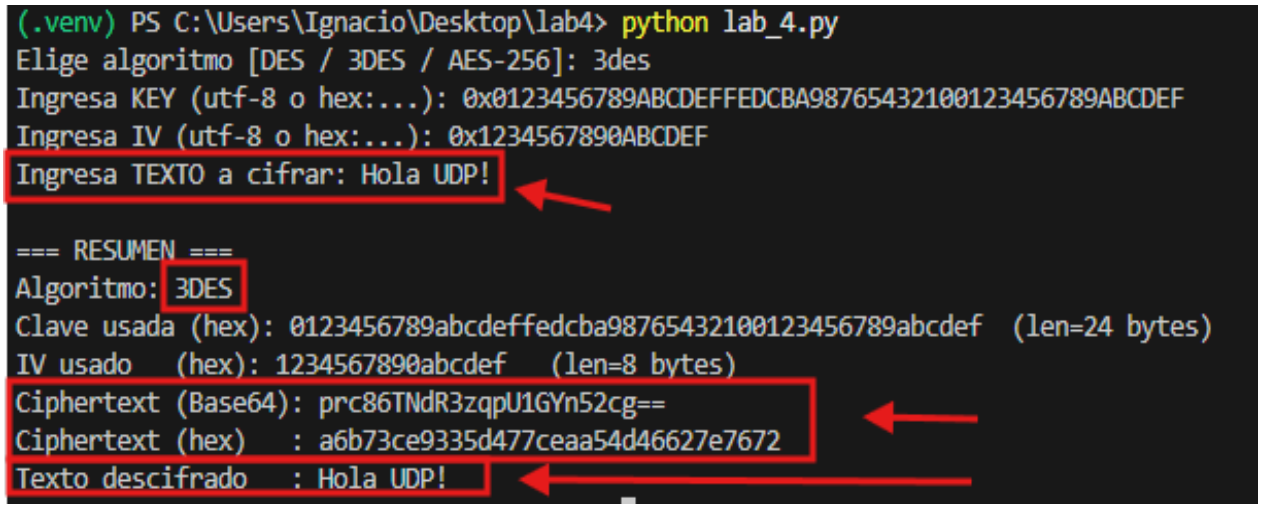

Figura 8: Cifrado y descifrado con el algoritmo 3DES en modo CBC.

La ejecución demuestra que el algoritmo 3DES en modo CBC cifra y descifra correctamente el texto “Hola UDP!”, generando un resultado diferente al cifrar y recuperando exactamente el mensaje original.

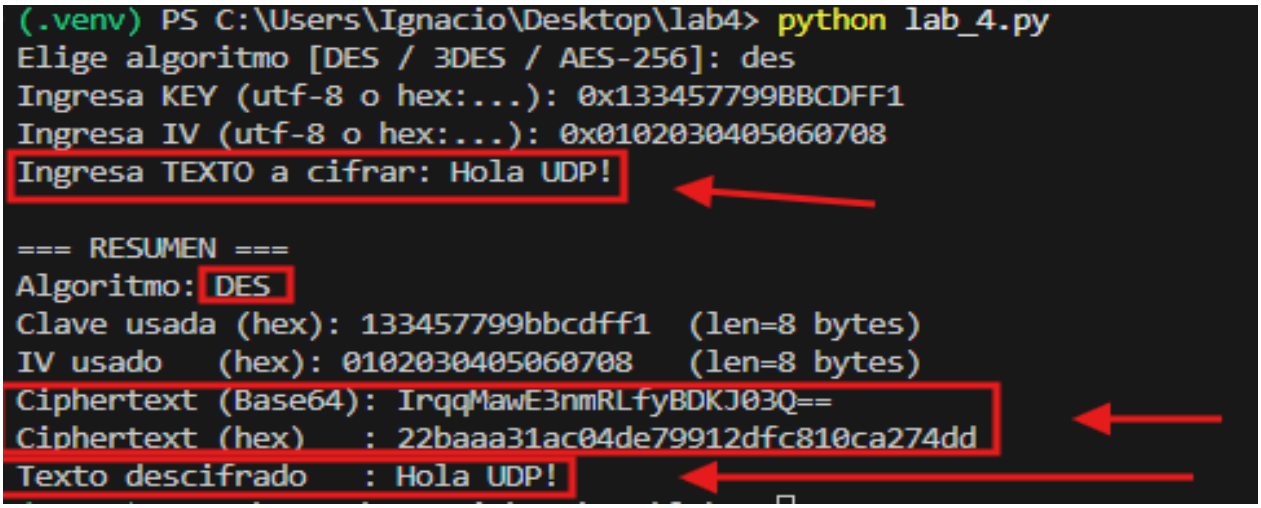

Figura 9: Cifrado y descifrado utilizando el algoritmo DES en modo CBC

En este caso, el algoritmo utiliza bloques y vectores de 8 bytes, obteniendo también un texto descifrado idéntico al original.

En los tres casos, el programa imprime la clave y el vector de inicialización utilizados, el texto cifrado en formato hexadecimal y Base64, y el texto descifrado. En todos los casos, el texto descifrado coincide con el original, lo que demuestra la correcta implementación de las funciones de cifrado y descifrado en modo CBC para cada algoritmo.

El comportamiento observado confirma además la sensibilidad del modo CBC, cualquier cambio en la clave o en el IV modifica completamente el resultado del texto cifrado, cumpliendo con los principios de difusión y aleatoriedad esperados en la criptografia simétrica.

2.5. Compara los resultados con un servicio de cifrado online

Para verificar la exactitud de la implementación, se compararon los resultados del programa con la página CyberChef. Se escogío el algoritmo **AES-256** en modo **CBC**, aplicando los mismos parámetros:

- **KEY:** 0x603deb1015ca71be2b73aef0857d77811f352c073b6108d72d9810a30914dff4
- **IV:** 0x000102030405060708090a0b0c0d0e0f
- **Texto:** Hola UDP!

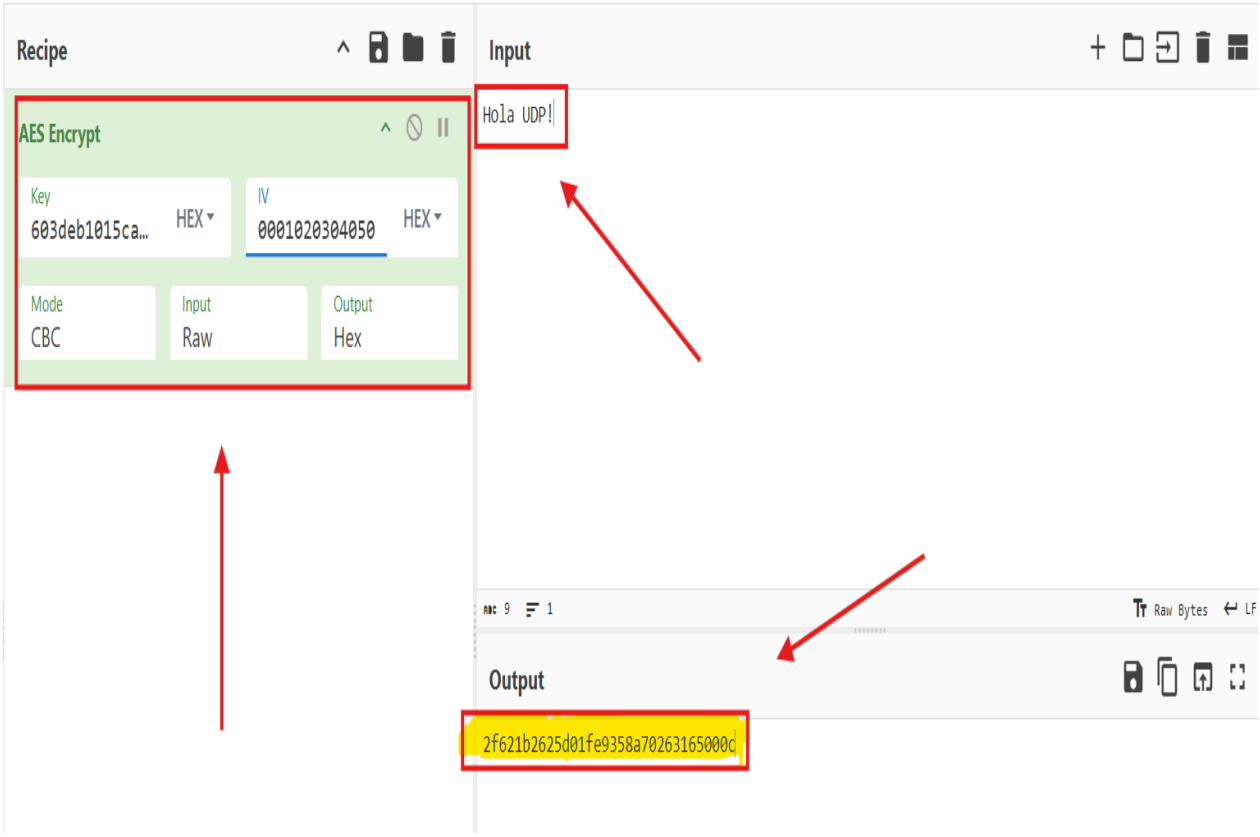

Figura 10: Resultado CyberChef utilizando AES-256.

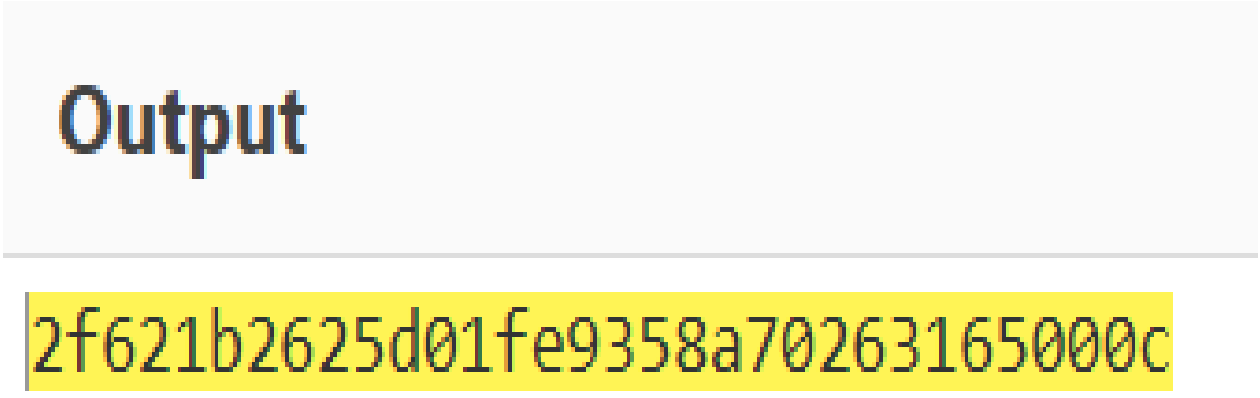

Figura 11: Resultado del cifrado en la herramienta CyberChef con los mismos parámetros.

Ambos sistemas (comparar Figuras 10 y 11 con la figura 7) generaron el mismo texto cifrado, tanto en formato hexadecimal (`2f621b2625d01fe9358a70263165000c`) como en

Base64 (`L2IbJiXQH+k1inAmMWUADA==`). Esto confirma que la implementación local reproduce exactamente el comportamiento del estándar **AES-256 en modo CBC** con relleno **PKCS7**. Con estos resultados se evidencia que el cifrado y decifrado funcionan correctamente.

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

El cifrado simétrico es ampliamente utilizado en la vida cotidiana para proteger información sensible que requiere confidencialidad y rapidez en el processamiento. Un caso concreto de aplicacion es la **transmisión segura de datos entre un cliente y un servidor** dentro de una red privada o una conexión VPN. En estos sistemas, ambos extremos comparten una misma clave de cifrado, lo que permite enviar información cifrada en tiempo real sin comprometer la velocidad.

Entre los algoritmos disponibles, se recomendaría utilizar **AES-256**, debido a que ofrece un excelente equilbrio entre seguridad y eficiencia computacionl. Su longitud de clave (256 bits) proporciona gran protección frente a ataques de fuerza bruta y es actualmente el estándar adoptado por organismos internacionales como el NIST.

Si la contraparte propusiera utilizar funciones **hash** en lugar de cifrado simétrico, se aclararía que los hashes (*SHA-256*, *MD5*, etc.) no son adecuados para este propósito, ya que son *unidireccionales* y no permiten recuperar el mensaje original. Mientras el cifrado simétrico busca preservar la confidencialidad del dato y permitir su descifrado posterior, las funciones hash están diseñadas para verificar integridad o autenticidad.

En conclusión, el uso de cfrado simétrico particularmente con AES-256— resulta ideal en contextos donde se necesita proteger información que deba ser posteriormente recuperada, garantizando tanto la confidencialidad como la eficiencia en la comunicación.

Conclusiones y comentarios

El desarrollo del laboratorio permitió comprender de forma práctica los principios del cifrado simétrico y la importancia de los parámetros que intervienen en su funcionamiento, como la clave (*key*) y el vector de inicialización (*IV*). Utilizando el Python se comprobó el comportamiento de los algoritmos DES, 3DES y AES-256, destacando las diferencias en longitud de clave, nivel de seguridad y eficiencia.

Durante el proceso se probaron diversas combinaciones con distintos valores y longitudes, comparando los resultados obtenidos con los generados por la página en línea **CiberChef**. Además, se ingresaron valores en formato **UTF-8**, verificándose que el programa interpretó correctamente los datos y mantuvo el resultado esperado.

Los resultados obtenidos demostraron que el programa cumple correctamente con el proceso de cifrado y descifrado en modo **CBC**, manteniendo la integridad del mensaje original tras el descifrado. Además,la herramienta online CyberChef confirmó q la implementación local es correcta, validando su precisión y aplicación del estándar PKCS7(padding).

Se observo que es fundamental ingresar todos los datos en el formato correcto, ya que cualquier variación en la clave, el IV o la codificación puede generar un resultado completamente

diferente.

Finalmente, se evidenció que el algoriitmo **AES-256** es el más recomendable por su alto nivel de seguridad, siendo adoptado internacionalmente como estándar moderno de cifrado. Esta experiencia permitió comprender no solo los conceptos teóricos, sino también su desarrollo práctico, reforzando el aprendizaje sobre la criptografía.