Informe Laboratorio 4

Sección 3

Isidora Bravo Ortiz
e-mail: isidora.bravo2@mail.udp.cl

Noviembre de 2025

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 4
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 10

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

En DES, la clave tiene 8 bytes (64 bits, de los cuales 56 son efectivos y 8 son de paridad) y el IV típico es de 8 bytes porque su bloque es de 64 bits.

En 3DES, la versión de dos claves (EDE2) usa 16 bytes (112 bits efectivos) y la de tres claves (EDE3) usa 24 bytes (168 bits efectivos); en ambos casos el IV sigue siendo de 8 bytes.

AES-256 emplea una clave de 32 bytes (256 bits) y, en modos como CBC/CFB/OFB/CTR, un IV de 16 bytes (bloque de 128 bits); en AES-GCM se recomienda un IV de 12 bytes (96 bits).

En cuanto a diferencias, DES está obsoleto por su baja seguridad; 3DES fue una mejora, pero hoy también se desaconseja por su lentitud y por trabajar con bloques de 64 bits, lo que favorece colisiones a gran escala; AES-256 es el estándar actual por su solidez y alto rendimiento (incluye aceleración por hardware).

Además, DES/3DES se basan en una red de Feistel, mientras que AES utiliza una red de sustitución-permuta (S-P) más eficiente. En todos los casos, el IV no es secreto, pero debe ser único (idealmente aleatorio) por mensaje; no debe reutilizarse con la misma clave en modos como CTR o GCM.

A continuación se adjunta una tabla con un resumen de la información mencionada anteriormente:

Tabla 1: Tamaño de clave e IV (en bytes)

Algoritmo	**Clave (bytes)**	**IV (bytes)**
DES	8	8
3DES (2 claves, EDE2)	16	8
3DES (3 claves, EDE3)	24	8
AES-256	32	16

2.2. Solicita datos de entrada desde la terminal

Para llevar a cabo el proceso de cifrado utilizando cualquiera de los tres algoritmos disponibles, es necesario solicitar al usuario los siguientes datos: el texto plano (mensaje a cifrar), la clave y el vector de inicialización (IV). Es importante destacar que tanto la clave como el vector de inicialización son requeridos de forma individual para cada uno de los algoritmos. A continuación, se presenta un ejemplo de cómo se visualiza este procedimiento en la terminal:

```
isidora@evilpancito:~/Descargas/Tarea2/doom$ /bin/python3 "/home/isidora/Descargas/Lab 4 cripto /lab4.py"
=== Sistema de Cifrado Simétrico (CBC) ===
Texto a cifrar: WenaWena

-- DES --
Key DES: MiClave1
IV DES (hex/base64/texto, 8 bytes): A1B2C3D4E5F60718

-- 3DES --
Key 3DES: ClaveTripleDES1234567890AB
IV 3DES (hex/base64/texto, 8 bytes): 1122334455667788

-- AES-256 --
Key AES-256: ClaveAES256bit_1234567890abcdef
IV AES (hex/base64/texto, 16 bytes): 00112233445566778899AABBCCDDEEFF
```

Figura 1: Solicitud de datos por terminal

2.3. Valida y ajusta la clave según el algoritmo

A continuación se muestra la salida del código, donde para cada algoritmo se presentan el texto ajustado para el cifrado, las claves utilizadas y el resultado del descifrado:

```
[DES] Resultados
 - Ajuste de clave  : sin cambios
 - Clave final (hex): 4D69436C61766531
 - IV (hex)         : A1B2C3D4E5F60718
 - Cifrado (hex)    : 927E11AA3A96E8F60C367C42272CEC61
 - Cifrado (b64)    : kn4RqjqW6PYMNnxCJyzsYQ==
 - Descifrado (utf8): WenaWena

[3DES] Resultados
 - Ajuste de clave  : truncado
 - Clave final (hex): 436C617665547269706C6544455331323334353637383930
 - IV (hex)         : 1122334455667788
 - Cifrado (hex)    : C4E65F327E8E7913F6392781B90F7C12
 - Cifrado (b64)    : xOZfMn6OeRP2OSeBuQ98Eg==
 - Descifrado (utf8): WenaWena

[AES-256] Resultados
 - Ajuste de clave  : relleno aleatorio
 - Clave final (hex): 436C6176654145533235366269745F31323334353637383930616263646566F3
 - IV (hex)         : 00112233445566778899AABBCCDDEEFF
 - Cifrado (hex)    : 64C0D76FBE8B9C5897F30DDB2E93D03F
 - Cifrado (b64)    : ZMDXb76LnFiX8w3bLpPQPw==
 - Descifrado (utf8): WenaWena
```

Figura 2: Resultado del cifrado

Como se puede apreciar, cada clase se encarga de ajustar la clave proporcionada por el usuario al tamaño adecuado y de validarla antes de emplearla en los procesos de cifrado y descifrado.

En el ejemplo mostrado, el usuario ingresó un mensaje junto con los valores correspondientes para cada algoritmo. En los 3 casos, se introdujeron dos entradas: una para la clave y otra para el vector de inicialización (IV).

Dado que algunos de estos valores no coincidían con las longitudes requeridas por cada algoritmo, el programa realizó los ajustes necesarios, ya fuera truncando los datos o completándolos

con bytes aleatorios. A pesar de estas modificaciones, los procesos de cifrado y descifrado se ejecutaron correctamente, permitiendo recuperar el mensaje original sin errores.

2.4. Implementa el cifrado y descifrado en modo CBC

Para cifrar y descifrar el mensaje proporcionado por el usuario, se usará el modo Cipher Block Chaining (CBC) para todos los algoritmos. Para esto utilizaremos el siguiente fragmento de código:

```python
# ---------- funciones por algoritmo (cifrar/descifrar) ----------
def cifrar_des(key: bytes, iv: bytes, pt: bytes) -> bytes:
    c = DES.new(key, DES.MODE_CBC, iv=iv)
    return c.encrypt(pad(pt, DES.block_size))

def descifrar_des(key: bytes, iv: bytes, ct: bytes) -> bytes:
    c = DES.new(key, DES.MODE_CBC, iv=iv)
    return unpad(c.decrypt(ct), DES.block_size)

def cifrar_3des(key: bytes, iv: bytes, pt: bytes) -> bytes:
    c = DES3.new(key, DES3.MODE_CBC, iv=iv)
    return c.encrypt(pad(pt, DES3.block_size))

def descifrar_3des(key: bytes, iv: bytes, ct: bytes) -> bytes:
    c = DES3.new(key, DES3.MODE_CBC, iv=iv)
    return unpad(c.decrypt(ct), DES3.block_size)

def cifrar_aes(key: bytes, iv: bytes, pt: bytes) -> bytes:
    c = AES.new(key, AES.MODE_CBC, iv=iv)
    return c.encrypt(pad(pt, AES.block_size))

def descifrar_aes(key: bytes, iv: bytes, ct: bytes) -> bytes:
    c = AES.new(key, AES.MODE_CBC, iv=iv)
    return unpad(c.decrypt(ct), AES.block_size)
```

Figura 3: Funciones de cifrado y descifrado

Como se puede apreciar, en cada algoritmo (DES, 3DES y AES) se aplica cifrado simétrico en modo CBC: el texto plano se rellena primero con padding PKCS7 para que su longitud sea múltiplo del tamaño de bloque (8 bytes en DES/3DES, 16 bytes en AES).

Luego, usando la misma clave y un vector de inicialización (IV) del tamaño exigido por el algoritmo, el modo CBC cifra bloque a bloque encadenando cada bloque de texto con el ciphertext anterior (el primero con el IV), lo que evita patrones repetidos y mejora la seguridad.

Para recuperar el mensaje, se realiza el proceso inverso con la misma clave y mismo IV: se descifra bloque a bloque y, al final, se elimina el padding. Si la clave o el IV no coinciden, o si los datos fueron alterados, el descifrado produce un padding inválido y el procedimiento falla, lo que también funciona como verificación básica de integridad.

Se puede comprobar que el cifrado y descifrado es un éxito, ya que se obtiene de vuelta la misma clave ingresada por el usuario, como se puede apreciar a continuación:

```
isidora@evilpancito:~/Descargas/Tarea2/doom$ /bin/python3 "/home/isidora/Descargas/Lab 4 cripto /lab4.py"
=== Sistema de Cifrado Simétrico (CBC) ===
Texto a cifrar: WenaWena
```

Figura 4: Ingreso de clave

```
[DES] Resultados
 - Ajuste de clave  : sin cambios
 - Clave final (hex): 4D69436C61766531
 - IV (hex)         : A1B2C3D4E5F60718
 - Cifrado (hex)    : 927E11AA3A96E8F60C367C42272CEC61
 - Cifrado (b64)    : kn4RqjqW6PYMNnxCJyzsYQ==
 - Descifrado (utf8): WenaWena

[3DES] Resultados
 - Ajuste de clave  : truncado
 - Clave final (hex): 436C617665547269706C6544455331323334353637383930
 - IV (hex)         : 1122334455667788
 - Cifrado (hex)    : C4E65F327E8E7913F6392781B90F7C12
 - Cifrado (b64)    : xOZfMn6OeRP2OSeBuQ98Eg==
 - Descifrado (utf8): WenaWena

[AES-256] Resultados
 - Ajuste de clave  : relleno aleatorio
 - Clave final (hex): 436C6176654145533235366269745F31323334353637383930616263646566F3
 - IV (hex)         : 00112233445566778899AABBCCDDEEFF
 - Cifrado (hex)    : 64C0D76FBE8B9C5897F30DDB2E93D03F
 - Cifrado (b64)    : ZMDXb76LnFiX8w3bLpPQPw==
 - Descifrado (utf8): WenaWena
```

Figura 5: Clave descifrada en cada algoritmo

2.5. Compara los resultados con un servicio de cifrado online

Para la comparación se usará la página web **www.anycript.com** para comparar su resultado con el resultado otorgado por el código en python. Para este caso se eligió el algoritmo DES, dando el siguiente resultado:

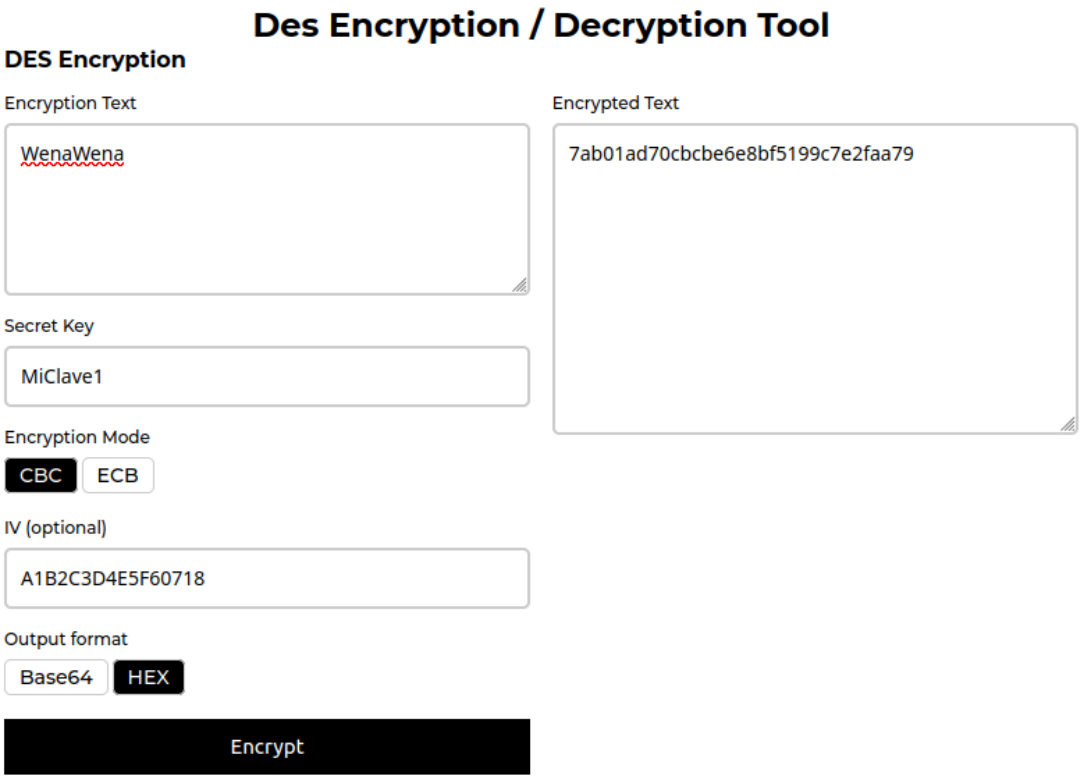

Figura 6: Cifrado DES en Anycript

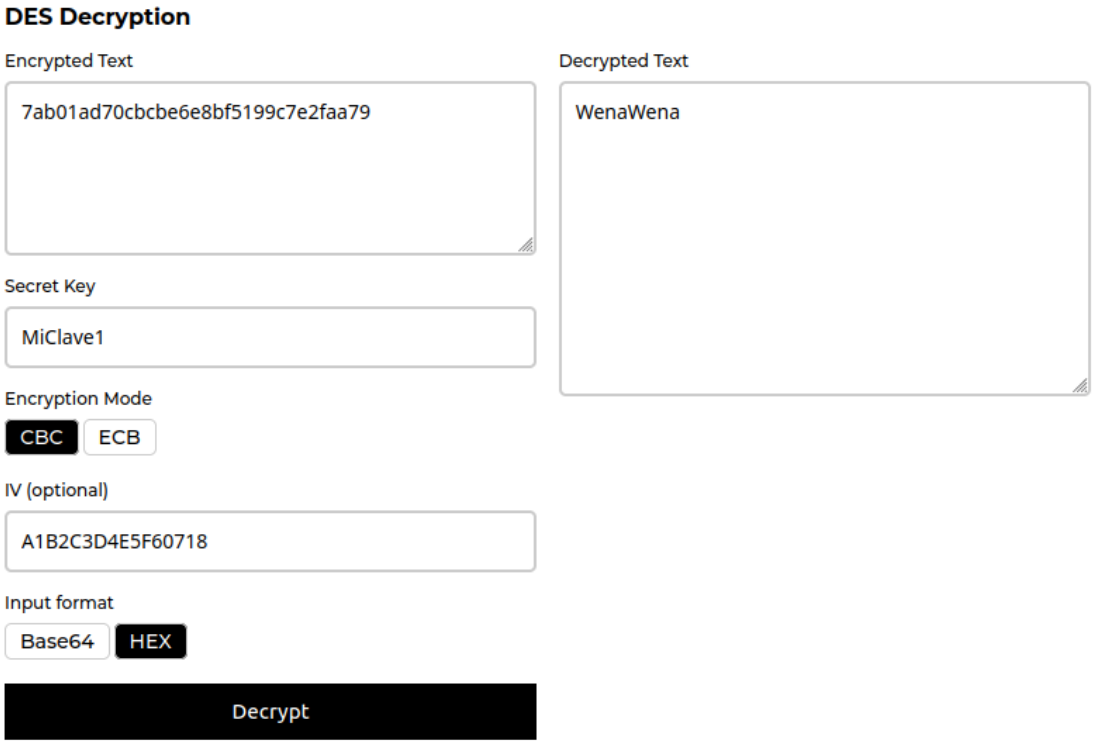

Figura 7: Descifrado DES en Anycript

```
[DES] Resultados
 - Ajuste de clave  : sin cambios
 - Clave final (hex): 4D69436C61766531
 - IV (hex)         : A1B2C3D4E5F60718
 - Cifrado (hex)    : 927E11AA3A96E8F60C367C42272CEC61
 - Cifrado (b64)    : kn4RqjqW6PYMNnxCJyzsYQ==
 - Descifrado (utf8): WenaWena
```

Figura 8: Cifrado y Descifrado DES código

Se empleó DES con el mismo texto y el mismo vector de inicialización en formato hexadecimal tanto en la página web como en Python; no obstante, los cifrados obtenidos difirieron, aunque en ambos casos fue posible recuperar correctamente el mensaje original.

Ello sugiere una diferencia en la interpretación o el tratamiento de la clave por parte de las herramientas comparadas. La explicación más plausible es que una implementación ajustó automáticamente los bits de paridad propios de DES y la otra no, o bien que una trató la clave como contraseña sujeta a transformación previa (derivación) mientras la otra utilizó la clave cruda sin modificaciones.

Dado que el mensaje “WenaWena” posee longitud de un bloque y que el tamaño del resultado

es consistente, el modo de operación y el esquema de relleno parecerían estar alineados; por lo tanto, la discrepancia se atribuye, con alta probabilidad, a diferencias en la clave efectiva empleada internamente.

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

El cifrado simétrico permite mantener información privada de forma rápida y eficiente y se aplica a archivos y discos de computadores, copias de seguridad, comunicaciones seguras en la web (el "candado"), conexiones remotas y datos en teléfonos o dispositivos conectados. Su mayor desafío es la llave compartida: debe generarse con buena entropía, guardarse en un lugar seguro (por ejemplo, un llavero/servicio de claves), rotarse periódicamente y limitar quién puede usarla. Cuando dos partes no comparten llave, primero acuerdan una de forma segura y luego cifran todo con ella.

Como caso ilustrativo, una app de salud que guarda fichas clínicas en el teléfono y respalda en la nube debe proteger los datos si el equipo se pierde o si la red es interceptada; allí conviene usar AES-256 con un modo que, además de ocultar, verifique que nada fue modificado (por ejemplo, GCM; si el equipo no acelera AES, ChaCha20-Poly1305 ofrece rendimiento similar). Si la contraparte propone "usar hashes en lugar de cifrado", corresponde aclarar que un hash solo permite comprobar cambios, no ocultar el contenido: no impide que terceros lean los datos ni permite revertirlos al texto original. En consecuencia, los hashes pueden complementar (integridad y control de versiones), pero no reemplazar el cifrado cuando el objetivo es confidencialidad.

Conclusiones y comentarios

En síntesis, el laboratorio logró implementar y validar cifrado simétrico (DES, 3DES y AES-256) en modo CBC con manejo correcto de padding, clave e IV, cumpliendo los requisitos de entrada, ajuste y verificación por descifrado. La comparación con un servicio online evidenció que, aun usando el mismo texto e IV, pequeñas diferencias en el tratamiento de la clave (por ejemplo, paridad de DES o derivaciones implícitas) pueden producir cifrados distintos sin comprometer la recuperación del mensaje, subrayando la necesidad de documentar con precisión cómo se forman las claves.

En términos de aplicabilidad, el trabajo refuerza que el cifrado simétrico es la opción práctica para proteger datos en reposo y en tránsito, y que conviene preferir variantes autenticadas (por ejemplo, AES-GCM) cuando sea posible. También deja como aprendizaje central que la seguridad no depende solo del algoritmo, sino de la gestión de llaves (generación, resguardo, rotación) y de la alineación exacta de parámetros entre sistemas. En conjunto, los resultados son correctos y reproducibles, y ofrecen una base sólida para integrar cifrado simétrico de manera responsable en aplicaciones reales.

Referencias

[1] A. J. Menezes, P. C. van Oorschot, S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.

[2] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 7th ed., Pearson, 2017.

[3] J. Katz, Y. Lindell, *Introduction to Modern Cryptography*, 3rd ed., CRC Press, 2020.