Informe Laboratorio 4

Sección 3

Diego Tapia
e-mail: diego.tapia6@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

DES, 3DES y AES-256 son algoritmos de cifrado simétrico y estos se diferencian en la longitud de sus claves y en el tamaño del vector de inicialización (IV), esto influye directamente en su seguridad.

En primer lugar, DES emplea bloques de 64 bits y una clave nominal de 64 bits, de los cuales solo 56 bits son efectivos y 8 se destinan a paridad. El IV en modo CBC debe ser también de 64 bits. Si bien fue un estándar ampliamente usado, hoy se considera inseguro por su vulnerabilidad frente a ataques de fuerza bruta [**stallings2017**].

Como alternativa, surge 3DES (Triple DES), que aplica el algoritmo DES tres veces con claves independientes. Su longitud total es de 192 bits (3×64), con una entropía efectiva de 168 bits y 24 de paridad. El tamaño del bloque y el IV permanecen en 64 bits. Aunque ofrece mayor seguridad que DES, su eficiencia es menor, ya que resulta aproximadamente tres veces más lento [**menezes1996**].

Finalmente, AES-256 utiliza un bloque de 128 bits y una clave de 256 bits efectivos, sin bits de paridad. En modo CBC, requiere un IV de 128 bits, acorde al tamaño del bloque. Además de realizar 14 rondas de sustitución y permutación, se ha consolidado como el estándar vigente en seguridad debido a su mayor resistencia y eficiencia en hardware y software [**daemen2002**].

Todo esto se resume en la siguiente tabla:

Tabla 1: Comparación de algoritmos de cifrado simétrico

Algoritmo	**Tamaño de clave**	**Tamaño del IV**
DES	8 bytes (64 bits, 56 efectivos)	8 bytes (64 bits)
3DES	24 bytes (3×8 bytes)	8 bytes (64 bits)
AES-256	32 bytes (256 bits)	16 bytes (128 bits)

2.2. Solicita datos de entrada desde la terminal

Para cifrar en cualquiera de los 3 algoritmos es necesario solicitar al usuario; el texto plano (mensaje a cifrar), la clave y el vector de inicialización (IV). Cabe mencionar que estos últimos dos se piden para cada uno de los algoritmos. A continuación se muestra como se ve a través de la terminal:

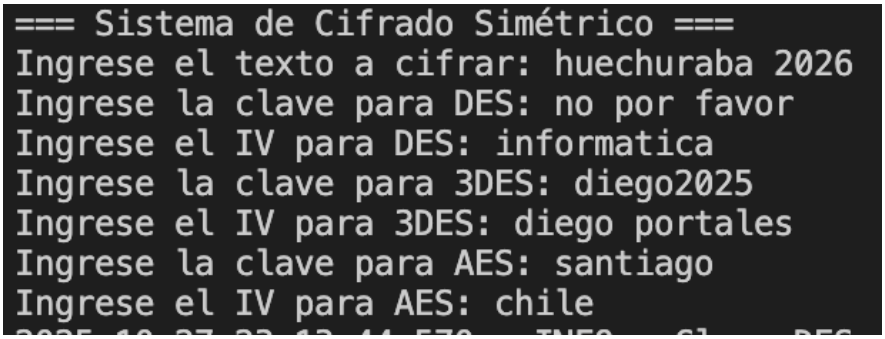

Figura 1: Solicitud de datos por consola

El resultado de la Figura 1 se debe al siguiente fragmento del código:

```python
class InterfazUsuario:
    """Maneja la interacción con el usuario."""

    @staticmethod
    def solicitar_datos() -> Dict[str, str]:
        """Solicita todos los datos necesarios al usuario."""
        datos = {}

        print("=== Sistema de Cifrado Simétrico ===")
        datos['mensaje'] = input("Ingrese el texto a cifrar: ")

        algoritmos = ['DES', '3DES', 'AES']
        for algoritmo in algoritmos:
            datos[f'clave_{algoritmo.lower()}'] = input(f"Ingrese la clave para {algoritmo}: ")
            datos[f'iv_{algoritmo.lower()}'] = input(f"Ingrese el IV para {algoritmo}: ")

        return datos
```

Figura 2: Interfaz del usuario

2.3. Valida y ajusta la clave según el algoritmo

A continuación se observa el output del código, en donde para cada algoritmo muestra como se ajusta el texto a cifrar, las llaves y el descifrado.

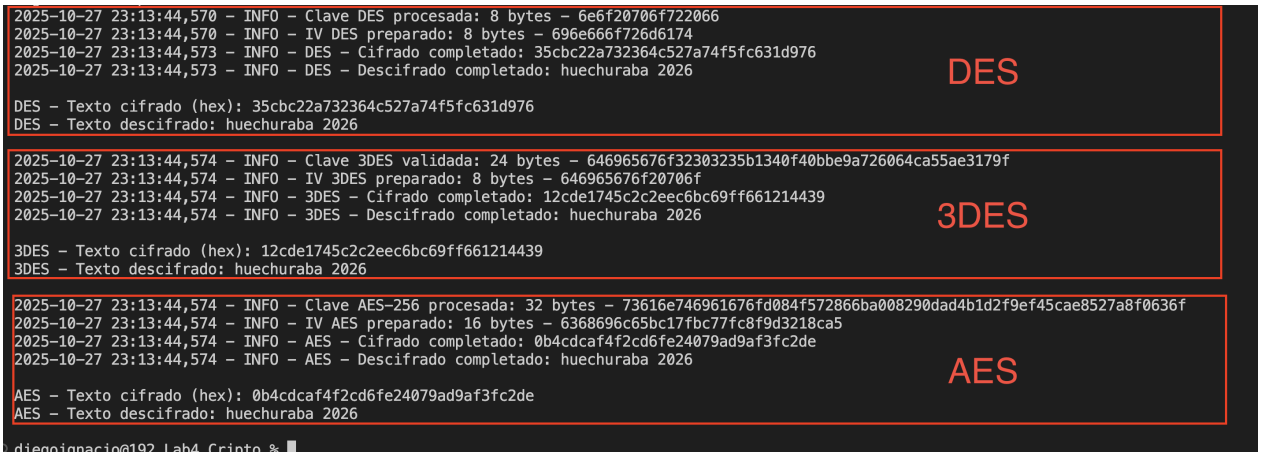

Figura 3: Output del los algoritmos

En la figura 4 se observa el extracto de código donde se ajustan las llaves.

```python
class ImplementacionDES(AlgoritmoCifrado):
    """Implementación específica para DES."""

    def preparar_clave(self, clave_texto: str) -> bytes:
        clave_bytes = self.procesador.normalizar_longitud(clave_texto, self.config.tamaño_clave)
        return self.validador.validar_clave_des(clave_bytes)

class Implementacion3DES(AlgoritmoCifrado):
    """Implementación específica para 3DES."""

    def preparar_clave(self, clave_texto: str) -> bytes:
        clave_bytes = self.procesador.normalizar_longitud(clave_texto, self.config.tamaño_clave)
        return self.validador.validar_clave_3des(clave_bytes)

class ImplementacionAES(AlgoritmoCifrado):
    """Implementación específica para AES-256."""

    def preparar_clave(self, clave_texto: str) -> bytes:
        clave_bytes = self.procesador.normalizar_longitud(clave_texto, self.config.tamaño_clave)
        return self.validador.validar_clave_aes(clave_bytes)
```

Figura 4: Ajuste de Clave

Cada clase ajusta la clave proporcionada por el usuario al tamaño correcto y la valida antes de usarla en el cifrado/descifrado.

En el ejemplo mostrado en la figura 3, el usuario ingresó un mensaje junto con los valores correspondientes para cada algoritmo. En el caso de DES se introdujo un input para la clave y otro para el vector de inicialización (IV). De manera similar, en 3DES se entregaron los

respectivos inputs de clave e IV, y finalmente para AES-256 se proporcionaron también su clave e IV. Dado que algunos de estos valores no coincidían con las longitudes requeridas por cada algoritmo, el programa aplicó los ajustes necesarios, ya sea truncando o completando con bytes aleatorios. A pesar de estas adaptaciones, el cifrado y descifrado se llevaron a cabo correctamente, logrando recuperar el mensaje original sin errores.

2.4. Implementación del cifrado y descifrado en modo CBC

El sistema desarrollado implementa los algoritmos simétricos DES, 3DES y AES bajo el modo Cipher Block Chaining (CBC). La configuración de cada algoritmo, incluyendo tamaño de clave, bloque, vector de inicialización (IV) y clase correspondiente, se define en la fábrica de algoritmos (Figura 5), donde se especifican los parámetros requeridos para cada caso.

```
class FabricaAlgoritmos:
    """Factory para crear instancias de algoritmos de cifrado."""

    CONFIGURACIONES = {
        'DES': ConfiguracionCifrado('DES', 8, 8, 8, DES),
        '3DES': ConfiguracionCifrado('3DES', 24, 8, 8, DES3),
        'AES': ConfiguracionCifrado('AES', 32, 16, 16, AES)
    }
```


Figura 5: Función clave para centralizar el código

Posteriormente, el método que ejecuta el cifrado completo es el encargado de realizar todo el proceso. En primer lugar, prepara la clave y el IV proporcionados por el usuario para ajustarlos a la longitud adecuada. Luego crea un objeto de cifrado inicializado en modo CBC, que permite cifrar el mensaje tras aplicar el padding necesario al texto plano (Figura 6). El resultado es el texto cifrado en formato hexadecimal.

De manera análoga, para el descifrado se instancia un nuevo objeto de descifrado, también configurado en modo CBC con la misma clave e IV. Con este objeto, el programa invierte la operación aplicada durante el cifrado, eliminando finalmente el padding para recuperar el mensaje original (Figura 6).

Finalmente, se imprime tanto el texto cifrado como el descifrado, lo que permite verificar explícitamente que el mensaje recuperado coincide con el texto original ingresado por el usuario. Esta comprobación asegura la correcta implementación del proceso de cifrado y descifrado.

```python
def ejecutar_cifrado_completo(self, mensaje: str, clave_texto: str, iv_texto: str) -> None:
    """Ejecuta el proceso completo de cifrado y descifrado."""
    clave = self.preparar_clave(clave_texto)
    iv = self.preparar_iv(iv_texto)

    # Crear cipher para cifrado
    cipher = self.config.clase_cipher.new(clave, self.config.clase_cipher.MODE_CBC, iv)

    # Cifrar mensaje
    mensaje_padded = pad(mensaje.encode('utf-8'), self.config.tamaño_bloque)
    texto_cifrado = cipher.encrypt(mensaje_padded)

    logger.info(f"{self.config.nombre} - Cifrado completado: {texto_cifrado.hex()}")

    # Crear nuevo cipher para descifrado
    decipher = self.config.clase_cipher.new(clave, self.config.clase_cipher.MODE_CBC, iv)

    # Descifrar mensaje
    texto_descifrado_bytes = decipher.decrypt(texto_cifrado)
    texto_descifrado = unpad(texto_descifrado_bytes, self.config.tamaño_bloque)

    logger.info(f"{self.config.nombre} - Descifrado completado: {texto_descifrado.decode('utf-8')}")
    print(f"\n{self.config.nombre} - Texto cifrado (hex): {texto_cifrado.hex()}")
    print(f"{self.config.nombre} - Texto descifrado: {texto_descifrado.decode('utf-8')}\n")
```

Figura 6: Función Cifrado y Descifrado

2.5. Compara los resultados con un servicio de cifrado online

Para realizar la validación, se utilizó el algoritmo DES en el programa implementado y se comparó con el servicio web AnyCript. En ambos casos se ingresó el mismo mensaje, la misma clave y el mismo vector de inicialización, manteniendo el modo de cifrado CBC.

En la Figura 8 se observa el proceso de cifrado en la plataforma online, mientras que la Figura 9 muestra la salida generada por el programa en Python. A pesar de que ambos entornos utilizan los mismos parámetros de entrada, el resultado del texto cifrado es distinto: la aplicación online entrega un resultado en formato Base64, mientras que el programa desarrollado presenta la salida en hexadecimal. Además, el servicio web no aplicó relleno (padding), mientras que en la implementación en Python se utilizó PKCS#7 para completar el bloque, lo que también contribuye a la diferencia en los valores.

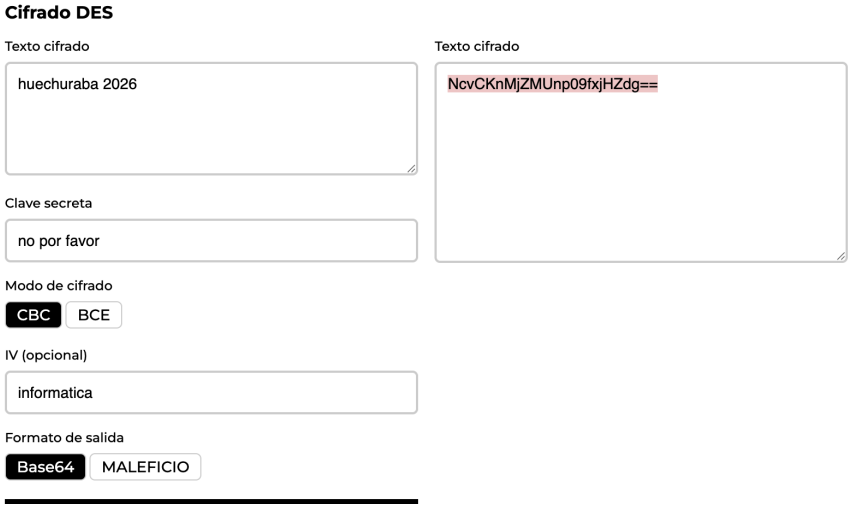

Figura 7: Cifrado DES realizado en la página web.

Figura 8: Decifrado DES realizado en la página web.

```
2025-10-27 23:13:44,570 - INFO - Clave DES procesada: 8 bytes - 6e6f20706f722066
2025-10-27 23:13:44,570 - INFO - IV DES preparado: 8 bytes - 696e666f726d6174
2025-10-27 23:13:44,573 - INFO - DES - Cifrado completado: 35cbc22a732364c527a74f5fc631d976
2025-10-27 23:13:44,573 - INFO - DES - Descifrado completado: huechuraba 2026
```

Figura 9: Resultado de cifrado y descifrado DES en simetricos.py.

A pesar de esta diferencia en la representación y el relleno aplicado, ambos sistemas logran descifrar el mensaje y recuperar el texto original correctamente, como se muestra en las figuras. Esto confirma que el funcionamiento del algoritmo es válido en ambos casos, y que

la variación del resultado cifrado se debe principalmente a diferencias de formato de salida y políticas de relleno.

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

El cifrado simétrico ha sido ampliamente utilizado en la protección de datos, aunque algunos algoritmos quedaron obsoletos con el tiempo. DES, por ejemplo, ofrecía solo 56 bits efectivos de seguridad, lo que lo hizo vulnerable a ataques de fuerza bruta, mientras que 3DES fue adoptado como una solución temporal, pero resultó demasiado lento y hoy ya no se recomienda. La transición hacia AES permitió establecer un nuevo estándar internacional, mucho más robusto y eficiente.

En la actualidad, AES-256 es el algoritmo de referencia en aplicaciones reales que van desde el cifrado de discos completos en notebooks, hasta la protección de comunicaciones en protocolos TLS y VPN, o el resguardo de respaldos de bases de datos en universidades y empresas. Su fortaleza radica está en su resistencia frente a ataques de fuerza bruta, en su eficiencia gracias a implementaciones optimizadas en hardware y software, y en su condición de estándar aceptado globalmente, lo que facilita la integración en sistemas modernos.

Finalmente DES y 3DES han quedado obsoletos en su uso criptográfico en la vida real, AES-256 hoy es la base de la seguridad simétrica en la vida real.

Conclusiones y comentarios

El desarrollo de este laboratorio permitió comprender en profundidad el funcionamiento de los algoritmos de cifrado simétrico DES, 3DES y AES-256, aplicados en modo CBC. A través de la implementación en Python se verificó que, a pesar de los ajustes necesarios en la longitud de claves e IVs, el proceso de cifrado y descifrado logra recuperar fielmente el mensaje original, lo que evidencia la correcta aplicación de los algoritmos.

Se constató además que, aunque el resultado cifrado difiere en comparación con servicios online debido al formato de salida (hexadecimal frente a Base64) y al uso de relleno (PKCS#7 frente a sin relleno), la integridad del descifrado es equivalente. Esto confirma que las variaciones en la representación no afectan la validez del algoritmo.

En términos de aplicabilidad, se observó que tanto DES como 3DES han quedado obsoletos por limitaciones de seguridad o rendimiento, mientras que AES-256 se mantiene como el estándar vigente. Este último constituye hoy la base de la seguridad en comunicaciones, almacenamiento y sistemas críticos.

En conclusión, la experiencia práctica no solo permitió afianzar conocimientos teóricos de criptografía, sino también evidenciar cómo las decisiones en la implementación —como la gestión de claves, el manejo del padding o la representación del cifrado— impactan directamente en el resultado final y en su uso en aplicaciones reales.

Referencias

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

[2] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.

[3] J. Daemen and V. Rijmen, *The Design of Rijndael: AES – The Advanced Encryption Standard*, Springer, 2002.