Informe Laboratorio 4

Sección 2

Matias Tobar
e-mail: matias.tobar@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 el desarrollo de este laboratorio se investigaron los tamaños de clave y vector de inicialización requeridos por cada algoritmo:

Algoritmo	Tamaño de Clave	Tamaño de IV	Tamaño de Bloque
DES	8 bytes (64 bits)	8 bytes (64 bits)	8 bytes
AES-256	32 bytes (256 bits)	16 bytes (128 bits)	16 bytes
3DES	24 bytes (192 bits)	8 bytes (64 bits)	8 bytes

Tabla 1: Tamaños de clave, IV y bloque para cada algoritmo

Principales diferencias:

- **DES:** Desarrollado en los años 70, utiliza claves de 56 bits efectivos. Es considerado inseguro por su vulnerabilidad a ataques de fuerza bruta y está descontinuado.
- **AES-256:** Estándar actual de cifrado simétrico, utiliza claves de 256 bits. Ofrece alta seguridad y excelente rendimiento, especialmente en hardware moderno con instrucciones AES-NI. Es el algoritmo recomendado para aplicaciones modernas.
- **3DES:** Mejora de DES que aplica el algoritmo tres veces consecutivas con tres subclaves. Más seguro que DES pero más lento. NIST lo deprecó en 2023 y está en proceso de descontinuación.

La información fue obtenida de la documentación de PyCryptodome y las especificaciones NIST (FIPS 197 para AES).

2.2. Solicita datos de entrada desde la terminal

El programa solicita tres datos desde la terminal: clave, vector de inicialización y texto a cifrar. Para la ejecución se utilizó un entorno virtual de Python (`venv`) debido a las políticas de gestión de paquetes del sistema.

Desarrollo del código:

El código fue generado con asistencia de la IA generativa Gemini de Google, proporcionando prompts específicos sobre los requisitos del laboratorio (algoritmos DES, AES-256 y 3DES en modo CBC, ajuste de claves e IVs, y uso de PyCryptodome).

Configuración del entorno:

Antes de ejecutar el programa, es necesario activar el entorno virtual e instalar las dependencias:

```
# Activar el entorno virtual
source venv/bin/activate

# Instalar la libreria pycryptodome
pip install pycryptodome

# Ejecutar el programa
python3 main.py
```

La implementación en `main.py` utiliza la función `input()`:

```
print("-" * 70)
clave_usuario = input("Ingrese la clave (key): ")
iv_usuario = input("Ingrese el vector de inicializacion (IV): ")
texto = input("Ingrese el texto a cifrar: ")
print("-" * 70)

texto_cifrado_des = ejecutar_des(clave_usuario, iv_usuario, texto)
texto_cifrado_aes = ejecutar_aes(clave_usuario, iv_usuario, texto)
texto_cifrado_3des = ejecutar_3des(clave_usuario, iv_usuario, texto)
```

Los mismos valores se utilizan para los tres algoritmos, permitiendo comparar directamente los resultados.

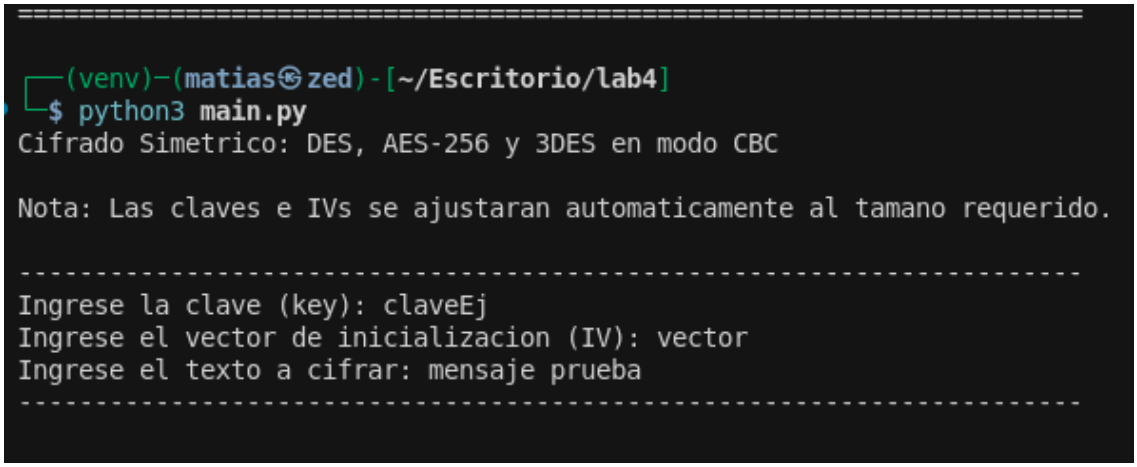

Figura 1: Solicitud de datos al usuario desde la terminal

2.3. Valida y ajusta la clave según el algoritmo

El programa implementa funciones que ajustan automáticamente las claves e IVs al tamaño requerido. Se utiliza `get_random_bytes()` de `Crypto.Random` para completar bytes faltantes:

```
from Crypto.Random import get_random_bytes

def ajustar_clave(clave_usuario, tamano_requerido, nombre_algoritmo):
    clave_bytes = clave_usuario.encode('utf-8')
    longitud_original = len(clave_bytes)

    if longitud_original < tamano_requerido:
        bytes_faltantes = tamano_requerido - longitud_original
        bytes_aleatorios = get_random_bytes(bytes_faltantes)
        clave_ajustada = clave_bytes + bytes_aleatorios
        print(f"Clave corta. Se agregaron {bytes_faltantes} bytes
    aleatorios.")
        print(f"Bytes aleatorios (hex): {binascii.hexlify(bytes_aleatorios
    ).decode()}")
    elif longitud_original > tamano_requerido:
        clave_ajustada = clave_bytes[:tamano_requerido]
        print(f"Clave larga. Se trunco a {tamano_requerido} bytes.")
    else:
        clave_ajustada = clave_bytes
        print("Clave tiene el tamano correcto.")

    print(f"Clave final (hex): {binascii.hexlify(clave_ajustada).decode()}
    ")
    return clave_ajustada
```

La función `get_random_bytes()` genera bytes criptográficamente seguros del sistema operativo. La clave final se muestra únicamente en formato hexadecimal para evitar problemas de representación con bytes no imprimibles. La función `ajustar_iv()` sigue la misma lógica.

A continuación se muestran ejemplos del ajuste de clave corta para cada algoritmo:

```
================================================================================
                              ALGORITMO DES
================================================================================

--- Ajuste de clave para DES ---
Tamano requerido: 8 bytes
Tamano ingresado: 7 bytes
Clave corta. Se agregaron 1 bytes aleatorios.
Bytes aleatorios (hex): 82
Clave final (hex): 636c617665456a82
```

Figura 2: Ajuste de clave corta para DES - Se completan bytes faltantes con aleatorios

```
======================================================================
                    ALGORITMO AES-256
======================================================================

--- Ajuste de clave para AES-256 ---
Tamano requerido: 32 bytes
Tamano ingresado: 7 bytes
Clave corta. Se agregaron 25 bytes aleatorios.
Bytes aleatorios (hex): fe36847fc6369f911a2798ba9acad3412a6f0bd41d6bed4717
Clave final (hex): 636c617665456afe36847fc6369f911a2798ba9acad3412a6f0bd41d6bed4717
```

Figura 3: Ajuste de clave corta para AES-256 - Se completan bytes faltantes con aleatorios

```
======================================================================
                    ALGORITMO 3DES
======================================================================

--- Ajuste de clave para 3DES ---
Tamano requerido: 24 bytes
Tamano ingresado: 7 bytes
Clave corta. Se agregaron 17 bytes aleatorios.
Bytes aleatorios (hex): 85bfee3ef4d7e1e0f4f353de93b6d4633d
Clave final (hex): 636c617665456a85bfee3ef4d7e1e0f4f353de93b6d4633d
```

Figura 4: Ajuste de clave corta para 3DES - Se completan bytes faltantes con aleatorios

2.4. Implementa el cifrado y descifrado en modo CBC

Se implementaron funciones de cifrado y descifrado para cada algoritmo utilizando modo CBC (Cipher Block Chaining). Este modo encadena los bloques cifrados usando XOR con el bloque anterior, siendo más seguro que ECB:

```python
from Crypto.Cipher import DES, AES, DES3
from Crypto.Util.Padding import pad, unpad

def cifrar_des(clave, iv, texto_plano):
    cipher = DES.new(clave, DES.MODE_CBC, iv)
    texto_bytes = texto_plano.encode('utf-8')
    texto_padded = pad(texto_bytes, DES.block_size)
    return cipher.encrypt(texto_padded)

def descifrar_des(clave, iv, texto_cifrado):
    cipher = DES.new(clave, DES.MODE_CBC, iv)
    texto_padded = cipher.decrypt(texto_cifrado)
    texto_bytes = unpad(texto_padded, DES.block_size)
    return texto_bytes.decode('utf-8')
```

El padding PKCS7 se aplica automáticamente para que el texto sea múltiplo del tamaño de bloque. Las funciones para AES-256 y 3DES tienen la misma estructura.

```
┌──(venv)─(matias㉿zed)-[~/Escritorio/lab4]
└─$ python3 main.py
Cifrado Simetrico: DES, AES-256 y 3DES en modo CBC

Nota: Las claves e IVs se ajustaran automaticamente al tamano requerido.

----------------------------------------------------------------------
Ingrese la clave (key): claveEj
Ingrese el vector de inicializacion (IV): vector
Ingrese el texto a cifrar: mensaje prueba
----------------------------------------------------------------------

======================================================================
                        ALGORITMO DES
======================================================================

--- Ajuste de clave para DES ---
Tamano requerido: 8 bytes
Tamano ingresado: 7 bytes
Clave corta. Se agregaron 1 bytes aleatorios.
Bytes aleatorios (hex): 82
Clave final (hex): 636c617665456a82

--- Ajuste de IV para DES ---
Tamano requerido: 8 bytes
Tamano ingresado: 6 bytes
IV corto. Se agregaron 2 bytes aleatorios.
Bytes aleatorios (hex): 0ac2
IV final (hex): 766563746f720ac2

--- Proceso de cifrado DES ---
Texto original: mensaje prueba
Texto cifrado (hex): 38126c8fdc88d1ca21dba273bc041023
Longitud: 16 bytes

--- Proceso de descifrado DES ---
Texto descifrado: mensaje prueba
Verificacion exitosa: Coincide con el original
```

Figura 5: Proceso de cifrado y descifrado con DES

```
========================================================================
                        ALGORITMO AES-256
========================================================================

--- Ajuste de clave para AES-256 ---
Tamano requerido: 32 bytes
Tamano ingresado: 7 bytes
Clave corta. Se agregaron 25 bytes aleatorios.
Bytes aleatorios (hex): fe36847fc6369f911a2798ba9acad3412a6f0bd41d6bed4717
Clave final (hex): 636c617665456afe36847fc6369f911a2798ba9acad3412a6f0bd41d6bed4717

--- Ajuste de IV para AES-256 ---
Tamano requerido: 16 bytes
Tamano ingresado: 6 bytes
IV corto. Se agregaron 10 bytes aleatorios.
Bytes aleatorios (hex): 5e0b0c96c1faa7e34840
IV final (hex): 766563746f725e0b0c96c1faa7e34840

--- Proceso de cifrado AES-256 ---
Texto original: mensaje prueba
Texto cifrado (hex): f625c973631e03d53f7c04385407a8c5
Longitud: 16 bytes

--- Proceso de descifrado AES-256 ---
Texto descifrado: mensaje prueba
Verificacion exitosa: Coincide con el original
```

Figura 6: Proceso de cifrado y descifrado con AES-256

```
================================================================================
                          ALGORITMO 3DES
================================================================================

--- Ajuste de clave para 3DES ---
Tamano requerido: 24 bytes
Tamano ingresado: 7 bytes
Clave corta. Se agregaron 17 bytes aleatorios.
Bytes aleatorios (hex): 85bfee3ef4d7e1e0f4f353de93b6d4633d
Clave final (hex): 636c617665456a85bfee3ef4d7e1e0f4f353de93b6d4633d

--- Ajuste de IV para 3DES ---
Tamano requerido: 8 bytes
Tamano ingresado: 6 bytes
IV corto. Se agregaron 2 bytes aleatorios.
Bytes aleatorios (hex): 2987
IV final (hex): 766563746f722987

--- Proceso de cifrado 3DES ---
Texto original: mensaje prueba
Texto cifrado (hex): 23ec496743619e6fc9ad32c49f846b51
Longitud: 16 bytes

--- Proceso de descifrado 3DES ---
Texto descifrado: mensaje prueba
Verificacion exitosa: Coincide con el original
```

Figura 7: Proceso de cifrado y descifrado con 3DES

En todas las ejecuciones se verifica que el texto descifrado coincide con el original, confirmando la correcta implementación.

```
================================================================================
                        RESUMEN DE RESULTADOS
================================================================================

Texto original: mensaje prueba

DES (hex):     38126c8fdc88d1ca21dba273bc041023
AES-256 (hex): f625c973631e03d53f7c04385407a8c5
3DES (hex):    23ec496743619e6fc9ad32c49f846b51

================================================================================
```

Figura 8: Resumen comparativo de los tres algoritmos

2.5. Compara los resultados con un servicio de cifrado online

Se seleccionó AES-256 para la comparación y se utilizó CyberChef, una herramienta online disponible en https://gchq.github.io/CyberChef.

El proceso fue el siguiente: primero se ejecutó el programa local anotando la clave e IV ajustados en formato hexadecimal. Estos valores se ingresaron exactamente en CyberChef configurado con AES en modo CBC.

```
========================================================================
                        ALGORITMO AES-256
========================================================================

--- Ajuste de clave para AES-256 ---
Tamano requerido: 32 bytes
Tamano ingresado: 7 bytes
Clave corta. Se agregaron 25 bytes aleatorios.
Bytes aleatorios (hex): fe36847fc6369f911a2798ba9acad3412a6f0bd41d6bed4717
Clave final (hex): 636c617665456afe36847fc6369f911a2798ba9acad3412a6f0bd41d6bed4717
```

Figura 9: Clave e IV ajustados del programa local para AES-256

```
========================================================================
                        ALGORITMO AES-256
========================================================================

--- Ajuste de clave para AES-256 ---
Tamano requerido: 32 bytes
Tamano ingresado: 7 bytes
Clave corta. Se agregaron 25 bytes aleatorios.
Bytes aleatorios (hex): fe36847fc6369f911a2798ba9acad3412a6f0bd41d6bed4717
Clave final (hex): 636c617665456afe36847fc6369f911a2798ba9acad3412a6f0bd41d6bed4717

--- Ajuste de IV para AES-256 ---
Tamano requerido: 16 bytes
Tamano ingresado: 6 bytes
IV corto. Se agregaron 10 bytes aleatorios.
Bytes aleatorios (hex): 5e0b0c96c1faa7e34840
IV final (hex): 766563746f725e0b0c96c1faa7e34840

--- Proceso de cifrado AES-256 ---
Texto original: mensaje prueba
Texto cifrado (hex): f625c973631e03d53f7c04385407a8c5
Longitud: 16 bytes

--- Proceso de descifrado AES-256 ---
Texto descifrado: mensaje prueba
Verificacion exitosa: Coincide con el original
```

Figura 10: Texto cifrado obtenido con el programa local

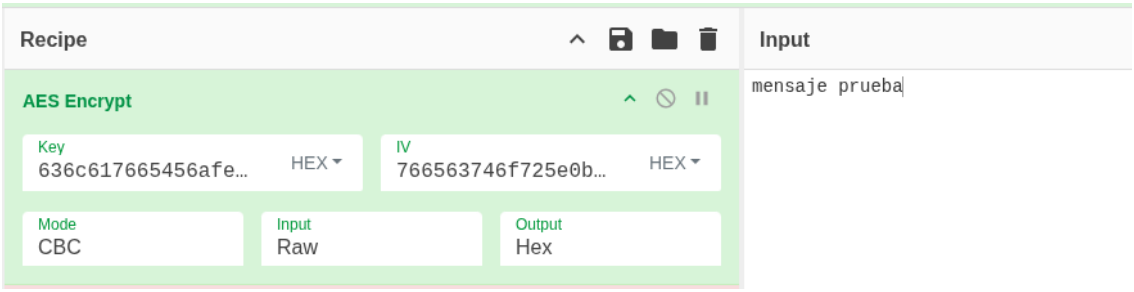

Figura 11: Configuración de CyberChef con los mismos parámetros

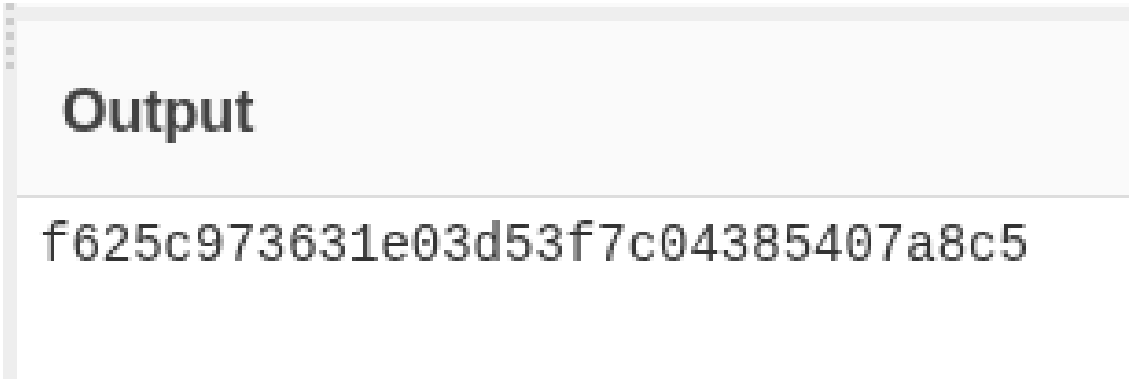

Figura 12: Resultado del cifrado en CyberChef: f625c973631e03d53f7c04385407a8c5

Los textos cifrados son idénticos (f625c973631e03d53f7c04385407a8c5). Esto valida que la implementación es correcta y cumple con el estándar AES-256. El cifrado simétrico es determinista: con la misma clave, IV y texto plano siempre produce el mismo resultado.

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

Caso de uso: Sistema de historias clínicas electrónicas

Un hospital necesita almacenar historias clínicas con información sensible (diagnósticos, tratamientos, resultados de exámenes) cumpliendo regulaciones como HIPAA. La solución es cifrar cada historia antes de almacenarla en la base de datos usando cifrado simétrico.

Algoritmo recomendado: AES-256

Se recomienda AES-256 en modo GCM por las siguientes razones:

- **Seguridad máxima:** Las claves de 256 bits son prácticamente inquebrantables con la tecnología actual. AES-256 está aprobado por NIST para proteger información clasificada como TOP SECRET del gobierno de Estados Unidos.
- **Rendimiento óptimo:** Los procesadores modernos incluyen instrucciones AES-NI (AES New Instructions) que aceleran significativamente las operaciones de cifrado y descifrado, permitiendo procesar grandes volúmenes de datos sin impacto perceptible en el rendimiento del sistema.

- **Cumplimiento normativo:** AES-256 cumple con los estándares requeridos por HIPAA (Health Insurance Portability and Accountability Act), GDPR (General Data Protection Regulation), ISO 27001 y PCI-DSS, facilitando las auditorías de seguridad.
- **Modo GCM (Galois/Counter Mode):** Este modo proporciona cifrado y autenticación simultáneamente (AEAD - Authenticated Encryption with Associated Data), lo que permite detectar cualquier modificación no autorizada de los datos cifrados, garantizando tanto confidencialidad como integridad.

Sobre usar hashes en lugar de cifrado:

Si la contraparte solicita implementar hashes en lugar de cifrado simétrico, mi respuesta sería que no es técnicamente viable para este caso de uso. Los hashes y el cifrado tienen propósitos fundamentalmente diferentes y no son intercambiables:

- **Cifrado simétrico:** Es un proceso reversible. Permite recuperar el dato original aplicando el proceso de descifrado con la clave correcta. Es ideal cuando necesitamos proteger información que debe ser consultada posteriormente.
- **Hashing:** Es un proceso irreversible por diseño. Una función hash toma un dato de entrada y produce un valor de longitud fija (digest) del cual es computacionalmente imposible recuperar el dato original.

En el contexto de historias clínicas, si aplicamos SHA-256 a una historia completa, obtendríamos un hash de 64 caracteres hexadecimales que no contiene ninguna información médica recuperable. No habría forma de que un médico consulte diagnósticos, tratamientos o resultados de exámenes. Sería equivalente a destruir permanentemente todas las historias clínicas del hospital.

Los hashes tienen aplicaciones específicas en seguridad:

- **Almacenamiento de contraseñas:** Se almacena el hash de la contraseña, no la contraseña misma. Durante la autenticación se compara el hash de la contraseña ingresada con el almacenado.
- **Verificación de integridad:** Se calcula el hash de un archivo para detectar si fue modificado, sin necesidad de recuperar el contenido original.
- **Firmas digitales:** Se firma el hash de un documento en lugar del documento completo por eficiencia.

En conclusión, hashing y cifrado son herramientas complementarias con propósitos distintos. Para proteger datos que deben ser consultados posteriormente, como historias clínicas, la única solución técnicamente viable es el cifrado simétrico. Los hashes no pueden sustituir al cifrado en este escenario.

Conclusiones y comentarios

Este laboratorio permitió implementar exitosamente tres algoritmos de cifrado simétrico (DES, AES-256 y 3DES) en modo CBC con Python. Se validó que el ajuste automático de claves e IVs funciona correctamente usando `get_random_bytes()` para completar y truncamiento para reducir.

La comparación con servicios online confirmó que la implementación es estándar e interoperable, produciendo resultados idénticos con los mismos parámetros. Esto demuestra el carácter determinista del cifrado simétrico y la correcta aplicación del modo CBC.

El análisis de aplicabilidad mostró que AES-256 es el algoritmo recomendado para aplicaciones modernas por su balance entre seguridad, rendimiento y cumplimiento normativo. También se aclaró la diferencia fundamental entre hashing y cifrado: el primero es irreversible y no puede sustituir al segundo cuando se requiere recuperar información.

- Repositorio de github: https://github.com/matiasts4/Lab-Criptografia