Informe Laboratorio 4

Sección 3

Sergio Soto Cuevas
e-mail: sergio.soto1@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

Tamaños de clave e IV requeridos para cada algoritmo.

Algoritmo	Clave (bytes)	IV (bytes)
DES	8	8
3DES	16 o 24	8
AES-256	32	16

Tabla 1: Longitudes requeridas de clave e IV.

Diferencias principales:

- DES: bloque de 64 bits, clave de 64 bits con 56 bits + 8 bits de paridad, realiza 16 rondas, el flujo es primero permutación inicial, 16 rondas Feistel y permutación final, obsoleto por la corta longitud de la key.[1]
- 3DES: esta usa DES aplicado tres veces con distintas claves cada vez, mantiene bloque de 64 bits, como mejora permite claves de 16 bytes o 24 bytes para mayor seguridad, por otro lado es mas lento comparado a DES dado que debe realizar 3 veces el cifrado.[2]
- AES-256: bloque fijo de 128 bits, clave de 256 bits, utiliza 14 rondas. Cada ronda transforma el estado matriz de 4×4 bytes mediante las operaciones SubBytes, ShiftRows, MixColumns y AddRoundKey. Se utiliza ampliamente y mejora en seguridad a 3DES y DES. [3]

[1] https://www.geeksforgeeks.org/computer-networks/data-encryption-standard-des-set-1
[2] https://www.geeksforgeeks.org/computer-networks/triple-des-3des
[3] https://www.geeksforgeeks.org/computer-networks/advanced-encryption-standard-aes

2.2. Solicita datos de entrada desde la terminal

Se utilizó la biblioteca PyCryptodome porque PyCrypto se encuentra obsoleta y presenta problemas de instalación.

```
import base64
from Crypto.Cipher import DES, AES, DES3
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad
```

Figura 1: Librerías utilizadas.

Los módulos importados son:

- base64: para representar el texto cifrado en Base64 al momento de imprimir en consola o registrar resultados.
- Crypto.Cipher (DES, AES, DES3): para instanciar los cifradores en modo CBC según el algoritmo requerido.
- Crypto.Random.get_random_bytes: para generar bytes aleatorios al completar longitud de claves o IV cuando corresponde.
- Crypto.Util.Padding (pad, unpad): para aplicar y retirar el padding conforme al tamaño de bloque del algoritmo.

Primero, se observa que en la Fig. 2 se solicita por consola el texto a cifrar para todos los algoritmos, , la clave y el vector de inicialización (IV) para cada algoritmo considerado: DES (CBC), 3DES (CBC) y AES-256 (CBC). Tras ingresar los datos, para cada algoritmo se muestra: la clave en hexadecimal, el IV en hexadecimal, el texto cifrado en Base64, el texto descifrado y una verificación que confirma si el descifrado coincide con el texto original.

```
Texto a cifrar(para todos):

KEY DES:
IV  DES:
DES (CBC)
clave en hex: 0651380f31a68052
IV en hex:  93d9ac39405bcbe2
Base64: Fc9qh9VXFFA=
Descifrado:
Verificación: esta bien
```

Figura 2: Consola. Ejemplo para DES

En la captura se aprecia el caso de DES (CBC), la clave en hex, el IV en hex, el Base64 correspondiente al cifrado, el Descifrado y la línea de Verificación, que indica si el proceso fue correcto.

A continuación se muestran main y la función correr_alg.

```python
def correr_alg(nombre: str, key_largo: int, iv_largo: int, key_in: str, iv_in: str, texto_in: str):
    key = ajustar_clave(key_in.encode("utf-8"), key_largo, nombre)
    iv  = ajustar_iv(iv_in.encode("utf-8"), iv_largo)
    texto_plano  = texto_in.encode("utf-8")
    texto_cifrado  = cifrar_cbc(nombre, key, iv, texto_plano)
    texto_decifrado  = descifrar_cbc(nombre, key, iv, texto_cifrado)

    print(f"{nombre} (CBC)")
    print(f"clave en hex: {key.hex()}")
    print(f"IV en hex:  {iv.hex()}")
    print(f"Base64: {base64.b64encode(texto_cifrado).decode()}")
    print(f"Descifrado: {texto_decifrado.decode('utf-8')}")
    print(f"Verificación:{' esta bien' if texto_decifrado == texto_plano else 'esta mal'}")

def main():
    texto = input("Texto a cifrar(para todos): ")
    des_key = input("\nKEY DES: ")
    des_iv  = input("IV  DES: ")
    correr_alg("DES", 8, 8, des_key, des_iv, texto)

    aes_key = input("\nKEY AES-256: ")
    aes_iv  = input("IV  AES-256: ")
    correr_alg("AES-256", 32, 16, aes_key, aes_iv, texto)

    tdes_key = input("\nKEY 3DES: ")
    tdes_iv  = input("IV  3DES: ")
    correr_alg("3DES", 24, 8, tdes_key, tdes_iv, texto)
```

Figura 3: Codigo main.

La función correr_alg recibe el nombre del algoritmo, las longitudes objetivo de clave e IV y las cadenas ingresadas por el usuario (clave, IV y texto). Esta ajuste los parámetros, prepara los datos y aplica cifrado y descifrado, por ultimo imprime el nombre del algoritmo, la clave y el IV en hexadecimal, el ciphertext en Base64, el texto descifrado en UTF-8 y una verificación que compara el descifrado con el texto original.

La función main implementa la interacción por consola y encadena la ejecución para los tres algoritmos. Solicita una vez el texto a cifrar para todos los algoritmos. Pide clave y IV para cada uno, por ultimo llama a correr_alg pasándole los parámetros correspondientes.

2.3. Valida y ajusta la clave según el algoritmo

Para que los tres algoritmos funcionen correctamente en modo CBC, se requiere:

- DES (CBC): clave de 8 bytes y IV de 8 bytes.
- 3DES (CBC): clave válida de 16 bytes o 24 bytes y IV de 8 bytes.
- AES-256 (CBC): clave de 32 bytes y IV de 16 bytes.

```python
def ajustar_clave(clave:bytes,largo:int,algoritmo:str) -> bytes:
    if len(clave) < largo:
        clave = clave + get_random_bytes(largo - len(clave))
    elif len(clave) > largo:
        clave = clave[:largo]

    if algoritmo == "3DES":
        if len(clave) not in (16, 24):
            clave = (clave + get_random_bytes(24 - len(clave)))[:24]
        clave = DES3.adjust_key_parity(clave)
    return clave
```

Figura 4: Validación y ajuste de clave.

La función de la Fig. 4 garantiza que la clave tenga el tamaño requerido por el algoritmo, esta recibe la clave ingresada, el largo objetivo y el nombre del algoritmo, y devuelve una clave con la longitud correcta:

1. Ajuste de longitud. Si la clave es más corta que el largo objetivo, se completa con la función get_random_bytes, si es más larga se trunca.

2. Para 3DES si la longitud ajustada no es 16 ni 24 bytes, se lleva hasta 24 bytes. Luego se corrige la paridad por byte con DES3.adjust_key_parity. DES y AES-256 no requieren ajuste de paridad, AES no lo usa, para DES PyCryptodome no lo exige.

```python
def ajustar_iv(iv: bytes, largo: int) -> bytes:
    if len(iv) < largo:
        iv = iv + get_random_bytes(largo - len(iv))
    elif len(iv) > largo:
        iv = iv[:largo]
    return iv
```

Figura 5: Ajuste de IV.

De misma forma se debe asegurar el largo correspondiente del IV. La función de la Fig. 5 realiza esto:

- Si el IV es más corto, se rellena con bytes aleatorios. Si es más largo, se trunca.

- Devuelve el IV final con longitud correcta, 8 bytes para DES/3DES y 16 bytes para AES-256.

2.4. Implementa el cifrado y descifrado en modo CBC

Las funciones cifrar_cbc y descifrar_cbc realizan el cifrado en modo CBC para los tres algoritmos.

```python
def cifrar_cbc(nombre: str, key: bytes, iv: bytes, texto: bytes) -> bytes:
    if nombre == "DES":
        return DES.new(key, DES.MODE_CBC, iv).encrypt(pad(texto, 8))
    if nombre == "AES-256":
        return AES.new(key, AES.MODE_CBC, iv).encrypt(pad(texto, 16))
    return DES3.new(key, DES3.MODE_CBC, iv).encrypt(pad(texto, 8))
```

Figura 6: Cifrado en modo CBC.

Cifrado (cifrar_cbc). Dado el nombre del algoritmo ("DES", .ªES-256.º "3DES"), se llama a PyCryptodome en modo CBC con la key y el iv. Luego, se aplica padding al texto plano al tamaño de bloque y se aplica encrypt, retornando los bytes del texto cifrado.

```python
def descifrar_cbc(nombre: str, key: bytes, iv: bytes, texto_cifrado: bytes) -> bytes:
    if nombre == "DES":
        return unpad(DES.new(key, DES.MODE_CBC, iv).decrypt(texto_cifrado), 8)
    if nombre == "AES-256":
        return unpad(AES.new(key, AES.MODE_CBC, iv).decrypt(texto_cifrado), 16)
    return unpad(DES3.new(key, DES3.MODE_CBC, iv).decrypt(texto_cifrado), 8)
```

Figura 7: Descifrado en modo CBC.

Descifrado (descifrar_cbc). Utiliza el mismo algoritmo, key y iv para llamar al objeto en modo CBC aplicando decrypt, se elimina el padding con unpad, esta función devuelve los bytes del texto descifrado.

2.5. Compara los resultados con un servicio de cifrado online

Ahora se comparara con un servicio de cifrado online, como lo es anycrypt.[4]
Para el algoritmo DES, se utilizara el texto cr7 , ademas se utilizaran claves y IV adecuados para no rellenar con bytes aleatorios.

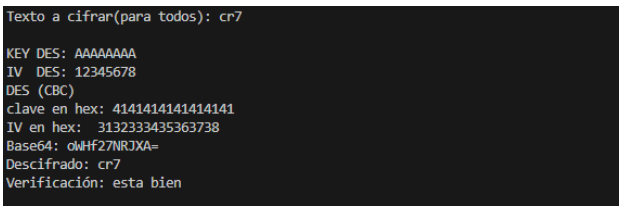

Figura 8: DES.

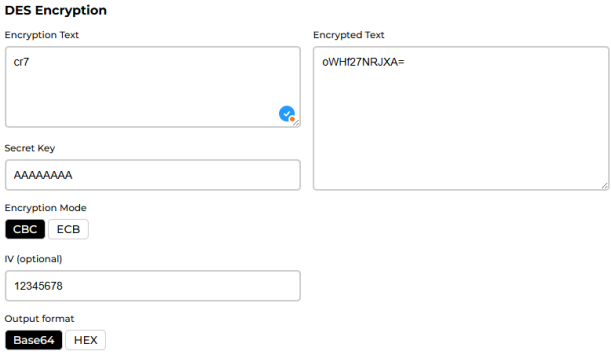

Figura 9: DES online.

Para comenzar se observa el el algoritmo DES, la salida tanto del algoritmo como del servicio online, son iguales correspondiendo a oWHf27NRJXA=.

A continuación se analizara que ocurre cuando la key y el iv, no corresponden a los que solicita cada algoritmo y como afectan los bytes aleatorios al cifrado. Se utilizaran como key penaldo y como IV pessi.

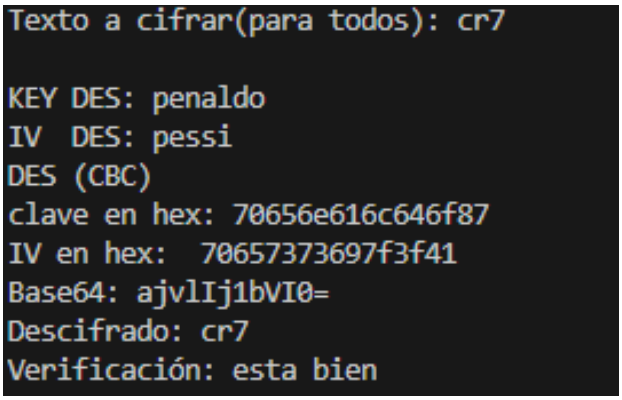

Figura 10: DES.

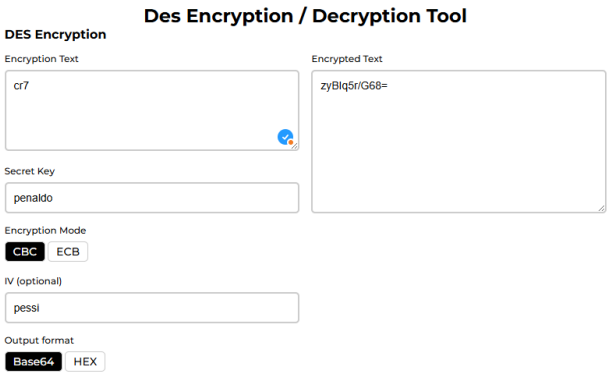

Figura 11: DES online.

Primero se observa que al utilizar estos parámetros en DES, tanto el algoritmo como el servicio online generan distintos outputs, debido a que el algoritmo utiliza bytes randoms para rellenar.Cabe destacar que la pagina utilizada no documenta el como rellena lo faltante, pero esto puede deberse a distinto padding aplicado ZeroPadding por ejemplo, o rellenar con

[4] https://anycript.com

alguna cadena no aleatoria puesto que independiente de la ejecución no cambia el cifrado en la web.

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

Un ejemplo de cuándo usar cifrado simétrico es al transmitir datos a través de una red interna segura. Por ejemplo, una empresa podría utilizar cifrado simétrico para proteger datos sensibles que se transfieren entre servidores dentro de su propia red. Este tipo de comunicación requiere rapidez y confidencialidad, pero el riesgo de que la clave sea interceptada es bajo debido al control interno de la infraestructura. Recomendaría el uso del algoritmo AES, ya que es actualmente es de los más seguros, eficientes para cifrado simétrico y por ello de los mas usados. Este tipo de algoritmos son más rápidos que los algoritmos asimétricos, esto los hace ideales para cifrar grandes volúmenes de datos dentro de una red interna. Ademas de que la clave se mantenga protegida, el cifrado simétrico proporciona un alto nivel de seguridad(siempre que esta red no sea vulnerada).

Si se sugiriera implementar funciones hash en lugar de cifrado simétrico, esto seria incorrecto puesto que no cumplen el mismo propósito, las funciones como SHA-256 o MD5, son irreversibles y son para verificar integridad o autenticación, el cifrado simétrico, permite cifrar como descifrar la información usando la misma clave, lo que es importante cuando se necesita recuperar los datos originales. Por lo tanto, sustituir cifrado simétrico por hashing sería incorrecto, ya que no se podria acceder nuevamente a la información cifrada, otra forma de verlo seria que se puede complementar el uso de las dos herramientas por ejemplo se podría cifrar con AES y agregar un HMAC para confirmar que el archivo no fue modificado.

Conclusiones y comentarios

En este laboratorio se implementó un programa que cifra y descifra en modo CBC usando DES, 3DES y AES-256, solicitando por consola texto, clave e IV, comprobándose el funcionamiento al comparar con un servicio online, se presentan diferencias aparecen cuando la clave/IV no cumplen longitud, porque se rellena con bytes aleatorios y la página no documenta su política de relleno.
En la vida real el cifrado simétrico es adecuado para proteger datos en tránsito dentro de una organización y se debe destacar que no cumple la misma función que un hash aunque estas herramientas podrían ser complementaria.

Referencias

[1] Sergio Soto. *Laboratorios Criptografía.* `https://github.com/SergioSoto1/LaboratoriosCriptografia`