Informe Laboratorio 4

Sección 1

Ignacio Santiago Medina Díaz
e-mail: ignacio.medina1@mail.udp.cl

Noviembre de 2024

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 comenzar, se investigó cada uno de los algoritmos a utilizar en el laboratorio (DES, 3DES y AES-256) utilizando las siguientes páginas:

- *https://www.pycryptodome.org/src/cipher/aes*
- *https://www.pycryptodome.org/src/cipher/des3module-Crypto.Cipher.DES3*
- *https://www.pycryptodome.org/src/cipher/desmodule-Crypto.Cipher.DES*

Según la información obtenida de las páginas anteriores, se recopilaron los siguientes datos para cada algoritmo.

- **Algoritmo DES**: Este algoritmo usa una **clave de 8 bytes (64 bits)**, de los cuales solo 56 bits son realmente empleados en el cifrado, mientras que los 8 bits restantes se utilizan para el control de paridad.
  - **Tamaño del IV**: **8 bytes (64 bits)**.
  - *Actualmente*, DES se considera inseguro debido a que su clave es relativamente corta, lo que lo expone a posibles ataques de fuerza bruta. Aunque su velocidad en hardware es alta, la limitada longitud de su clave restringe su aplicación en sistemas modernos.
- **Algoritmo 3DES**: Utiliza una **clave de 24 bytes (192 bits)**, compuesta por tres subclaves de 8 bytes cada una.
  - **Tamaño del IV**: **8 bytes (64 bits)**.
  - Este algoritmo es más seguro que DES al realizar el cifrado tres veces, pero, en comparación con AES, es menos seguro y eficiente. Además, su triple cifrado reduce su velocidad, lo cual limita su idoneidad para aplicaciones modernas.
- **Algoritmo AES-256**: Este método emplea una **clave de 32 bytes (256 bits)**.
  - **Tamaño del IV**: **16 bytes (128 bits)**.
  - AES-256 es uno de los algoritmos de cifrado más robustos y ampliamente usados hoy en día, siendo resistente a ataques de fuerza bruta. Además, es rápido y eficaz tanto en hardware como en software, y está optimizado para múltiples plataformas.

2.2. Solicita datos de entrada desde la terminal

Para este apartado del laboratorio se creó el siguiente programa a través del lenguaje de programación Python, utilizando la librería ***Crypto.Cipher import DES, 3DES, AES***, junto con ***Crypto.Util.Padding import pad, unpad*** y finalmente ***import base64***. El código final es el siguiente:

crypto.py > ...

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

def encrypt_message(algorithm, plaintext):
    # Establecer tamaño de clave y algoritmo
    if algorithm == 'DES':
        key_size = 8       # Clave de 8 bytes para DES
        iv_size = 8
        mode = DES.MODE_ECB

    elif algorithm == 'AES-256':
        key_size = 32      # Clave de 32 bytes para AES-256
        iv_size = 16       # AES requiere un IV de 16 bytes
        mode = AES.MODE_ECB

    elif algorithm == '3DES':
        key_size = 24      # Clave de 24 bytes para 3DES
        iv_size = 8
        mode = DES3.MODE_ECB

    else:
        print("Algoritmo no reconocido.")
        return

    # Pedir al usuario clave, IV y mensaje para cifrar
    key = input(f"Ingrese una clave de {key_size} bytes para {algorithm}: ").encode()

    # Validar que la clave tenga la longitud correcta
    if len(key) != key_size:
        print(f"La clave debe ser de {key_size} bytes.")
        return

    # Configuración de cifrado según el algoritmo seleccionado
    if algorithm == 'DES':
        cipher = DES.new(key, mode)
    elif algorithm == 'AES-256':
        cipher = AES.new(key, mode)
    elif algorithm == '3DES':
        cipher = DES3.new(key, mode)

    # Cifrar el mensaje
    padded_text = pad(plaintext, cipher.block_size)  # Ajustar al tamaño de bloque
    encrypted_data = cipher.encrypt(padded_text)
    encoded_data = base64.b64encode(encrypted_data).decode()

    # Mostrar el texto cifrado en base64
    print(f"\nMensaje cifrado (base64): {encoded_data}")

# Llamada a la función principal
if __name__ == "__main__":
    plaintext = input("Ingrese el mensaje a cifrar: ").encode()
    algorithm = input("Seleccione el algoritmo (DES, 3DES, AES-256): ")
    encrypt_message(algorithm, plaintext)
```

Figura 1: Código cifrado simetrico simple.

Con respecto al codigo de la figura 1, la función principal y más importante es la función `encrypt_message` que recibe dos parámetros: el `algorithm` (*el nombre del algoritmo de cifrado a usar*) y `plaintext` (*el mensaje en texto plano a cifrar*).

Cabe mencionar que a lo largo del laboratorio se utilizara el siguiente mensaje para cada uno de los algoritmos.

- ***Este es un mensaje secreto de prueba para el laboratorio de criptografía!***

A continuación se ejecutará el código anterior donde solicitara el mensaje a cifrar y el algoritmo a seleccionar.

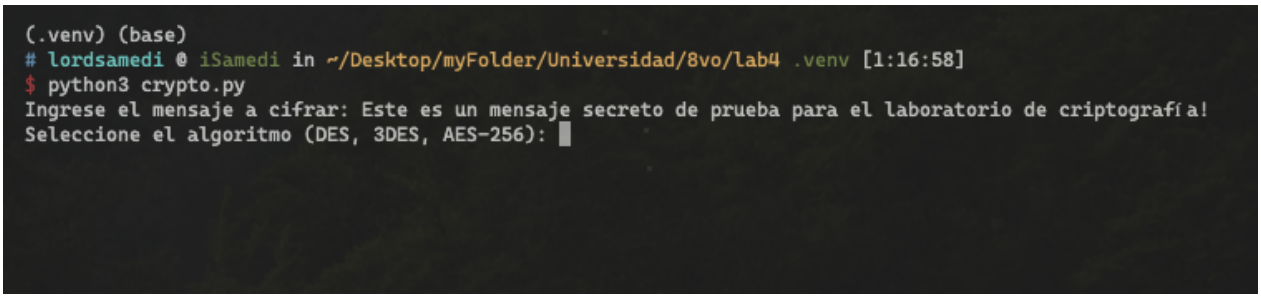

Figura 2: Código simple consola.

Al seleccionar el algoritmo se verifica cuál algoritmo ha seleccionado el usuario (`DES`, `AES-256`, o `3DES`).
Para cada algoritmo se especifica el tamaño de la clave (`key_size`) y el tamaño del vector de inicialización (`iv_size`) pero cabe mencionar que al elegir el **modo ECB, no requiere un vector de inicialización** (*unicamente se va a utilizar cuando se utilice el modo CBC*).

Al escoger el algoritmo ***DES*** (*anteriormente haber colocado el mensaje mencionado anteriormente*) se utilizo la clave **12345678** de **8 bytes** para obtener el siguiente cifrado DES fue "zECcTgVl3bWr/lczk1i4EddJrUZevCuQ9KAMJVBurRAezv3V1rA+IADdIn/PG3ypqQv/L4SF+qVpeO8HftCIGV04b1NXGLSdAF4ZuJbSHOE=". Tal como se muestrá en la siguiente imagen.

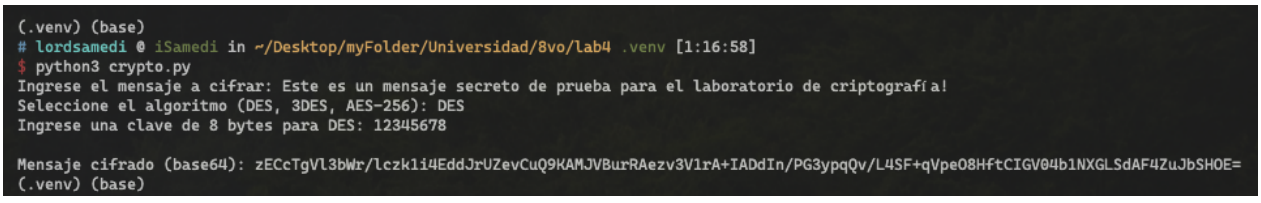

Figura 3: Output Algoritmo DES.

Al escoger el segundo algoritmo llamado ***3DES*** (*con utilizando el mensaje predeterminado*), se utilizo la clave **TripleDESde24bytesExacto** con **24 bytes justos**, ya que, el código no presenta una ajuste que pueda truncar o completa con bytes aleatorios si es muy corta. Finalmente, el cifrado fue "WSIRTo1gO9zcGTN6XpXwXwAvw2x6Rgda0MyFtvTNECYBoPF3bkrFBcgIIYrA+Pfy5rnvHtwb48KV2AdashO0Zp5+jkFxzEpaUWd6wEUysnM=". Tal como se muestrá en la siguiente imagen.

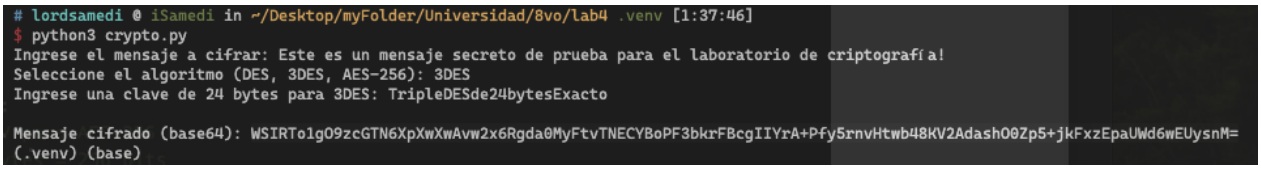

Figura 4: Output Algoritmo 3DES.

Finalmente, el ultimo algoritmo llamado ***AES-256***, en este caso se utilizó la clave de **32 bytes** fue `Esta_es_una_clave_AES_de_32_byte` , obteniendo la siguiente salida de texto cifrado: "gYsI3HTAglGvpS94Xx9F1VETRdMede7O02lD4kYBk0WHaUYfAjxX/JGMTRd7oE2LDy1FDBLAO/sC1yPFNR3C0tTCwgxZu9ngOuXSvoxSjTg=". Tal como se muestrá en la siguiente imagen.

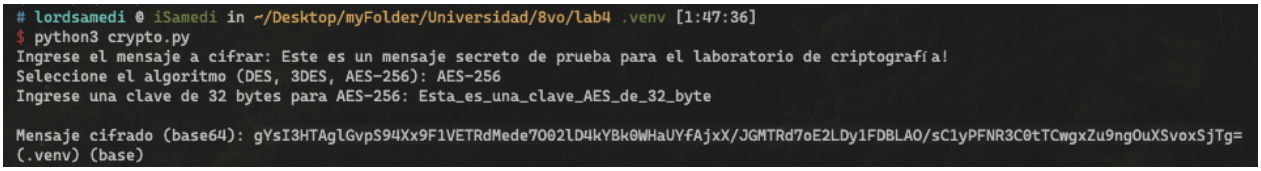

Figura 5: Output Algoritmo AES-256.

2.3. Valida y ajusta la clave según el algoritmo

Para este nuevo apartado, se ingresarán claves menores y mayores que el tamaño necesario. En el caso de claves menores, el algoritmo completará los bytes faltantes. En el caso de claves mayores, el algoritmo truncará la clave a la longitud requerida.

Para el caso de una clave menor al tamaño requerido, se utilizará una clave de 7 bytes para el algoritmo DES, que requiere una clave de 8 bytes. Para completar los bytes faltantes, se utilizó la siguiente función:

```
6   def adjust_key(key, key_size):
7       # Si la clave es más corta, se completa con bytes aleatorios
8       if len(key) < key_size:
9           key += get_random_bytes(key_size - len(key))
10      # Si la clave es más larga, se trunca a la longitud requerida
11      elif len(key) > key_size:
12          key = key[:key_size]
13      return key
14
```

Figura 6: Función para ajustar la clave según el algoritmo.

Para realizar el ajuste y validar la clave en función de la longitud requerida, se importo `get_random_bytes` de la biblioteca `Crypto.Random`. La función `adjust_key` (*función de la figura 6*)se encarga de ajustar la longitud de una clave (*key*) para que coincida con el tamaño especificado (`key_size`). Para lograrlo, primero verifica si la longitud de la clave es menor o mayor al tamaño requerido:

1. **Si la clave es más corta**: La función genera los bytes faltantes de manera aleatoria utilizando `get_random_bytes` y los agrega al final de la clave. Esto asegura que la clave alcance la longitud exacta necesaria.

2. **Si la clave es más larga**: La función trunca la clave, es decir, corta los bytes adicionales para reducirla a la longitud exacta necesaria.

Finalmente, la función devuelve la clave ajustada, garantizando que siempre tenga la longitud adecuada especificada en `key_size`.

Al ejecutar el nuevo código utilizando el **algoritmo DES** (***conciderando el tamaño necesario de 8 bytes para sus claves***), se utilizarón las siguientes claves:

1. **Clave corta**: *123456* (6 bytes).

2. **Clave larga**: *1234567890* (10 bytes).

Se obtuvieron los siguientes output al ejecutar el código con las claves mencionadas.

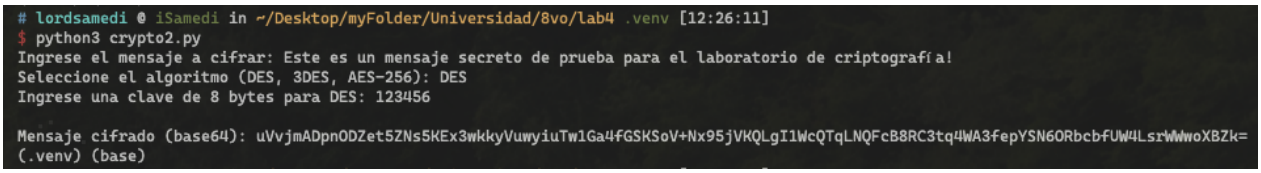

Figura 7: Output para Clave más Corta, DES.

Como se parecia en la figura 7, la función `adjust_key` funciona correctamente, en este caso se genero 2 bytes aleatorios para completar los 8 bytes de tamaño necesario para el algoritmo DES, obteniendo el siguiente mensaje cifrado en base64: üVvjmADpnODZet5ZNs5KEx3wkky VuwyiuTw1Ga4fGSKSoV+Nx95jVKQLgI1WcQTqLNQFcB8RC3tq4WA3fepYSN6ORbcbfU W4LsrWWwoXBZk=".

Utilizando la clave más larga, en este caso contando con 2 bytes adicionales, al truncarlo, se obtuvo el siguiente mensaje cifrado en base64 (tal como se muestra en la figura 8): "zEC-cTgVl3bWr/lczk1i4EddJrUZevCuQ9KAMJVBurRAezv3V1rA+IADdIn/PG3ypqQv/L4SF+ qVpeO8HftCIGV04b1NXGLSdAF4ZuJbSHOE=".

```
# lordsamedi @ iSamedi in ~/Desktop/myFolder/Universidad/8vo/lab4 .venv [12:34:26]
$ python3 crypto2.py
Ingrese el mensaje a cifrar: Este es un mensaje secreto de prueba para el laboratorio de criptografía!
Seleccione el algoritmo (DES, 3DES, AES-256): DES
Ingrese una clave de 8 bytes para DES: 1234567890

Mensaje cifrado (base64): zECcTgVl3bWr/lczk1i4EddJrUZevCuQ9KAMJVBurRAezv3V1rA+IADdIn/PG3ypqQv/L4SF+qVpeO8HftCIGV04b1NXGLSdAF4ZuJbSHOE=
(.venv) (base)
```

Figura 8: Output para Clave más Larga, DES.

Analizando el mensaje cifrado de la figura 8 y comparándolo con el mensaje cifrado con 8 bytes de tamaño (figura 3), se observa que los mensajes cifrados son idénticos. Esto concluye que la función `adjust_key` está funcionando correctamente, ya que en este caso truncó exítosamente la clave a los 8 primeros bytes, igualándose a la clave del apartado anterior.

Por otro lado, las claves utilizadas para el **algoritmo 3DES** (***tamaño necesario de 24 bytes***) son la siguientes:

1. **Clave corta**: *TripleDESde24bytesExac* (22 bytes).
2. **Clave larga**: *TripleDESde24bytesExactos!* (26 bytes).

A la hora de ejecutar el programa con las claves mencionadas se obtuvieron lo siguiente:

```
# lordsamedi @ iSamedi in ~/Desktop/myFolder/Universidad/8vo/lab4 .venv [12:34:42]
$ python3 crypto2.py
Ingrese el mensaje a cifrar: Este es un mensaje secreto de prueba para el laboratorio de criptografía!
Seleccione el algoritmo (DES, 3DES, AES-256): 3DES
Ingrese una clave de 24 bytes para 3DES: TripleDESde24bytesExac

Mensaje cifrado (base64): 8kapu2hSHb1kdcuV82X39pZTIgFFEolui7D6+Af9XDiX6HCPJmMd8Odc1UFWShKlwjHNhDmPr4Od5/nw0uk5X8yTW99dKICNn0JrXlNPDZE=
(.venv) (base)
```

Figura 9: Output para Clave más Corta, 3DES.

```
# lordsamedi @ iSamedi in ~/Desktop/myFolder/Universidad/8vo/lab4 .venv [13:21:45]
$ python3 crypto2.py
Ingrese el mensaje a cifrar: Este es un mensaje secreto de prueba para el laboratorio de criptografía!
Seleccione el algoritmo (DES, 3DES, AES-256): 3DES
Ingrese una clave de 24 bytes para 3DES: TripleDESde24bytesExactos!

Mensaje cifrado (base64): WSIRTo1gO9zcGTN6XpXwXwAvw2x6Rgda0MyFtvTNECYBoPF3bkrFBcgIIYrA+Pfy5rnvHtwb48KV2AdashO0Zp5+jkFxzEpaUWd6wEUysnM=
(.venv) (base)
```

Figura 10: Output para Clave más Larga, 3DES.

Igual que en el caso anterior, y repitiéndose con el algoritmo AES-256, al utilizar la misma clave pero con valores adicionales, estos se truncan e igualan a la clave del primer apartado (figura 4), igualándose los mensajes cifrados en base64.

Finalmente las claves para el **algoritmo AES-256** (***tamaño necesario 32 bytes***) son las siguientes para este apartado:

1. **Clave corta**: `Esta_es_una_clave_AES_de_32_bytes!` (34 bytes).
2. **Clave larga**: `Esta_es_una_clave_AES_de_32_by` (30 bytes).

Ejecutando el programa con las claves mencionadas se obtuvieron los siguientes outputs:

```
# lordsamedi @ iSamedi in ~/Desktop/myFolder/Universidad/8vo/lab4 .venv [13:29:44]
$ python3 crypto2.py
Ingrese el mensaje a cifrar: Este es un mensaje secreto de prueba para el laboratorio de criptografía!
Seleccione el algoritmo (DES, 3DES, AES-256): AES-256
Ingrese una clave de 32 bytes para AES-256: Esta_es_una_clave_AES_de_32_bytes!

Mensaje cifrado (base64): gYsI3HTAglGvpS94Xx9F1VETRdMede7O02lD4kYBk0WHaUYfAjxX/JGMTRd7oE2LDy1FDBLAO/sC1yPFNR3C0tTCwgxZu9ngOuXSvoxSjTg=
(.venv) (base)
```

Figura 11: Output para Clave más Corta, AES-256.

```
# lordsamedi @ iSamedi in ~/Desktop/myFolder/Universidad/8vo/lab4 .venv [13:30:11]
$ python3 crypto2.py
Ingrese el mensaje a cifrar: Este es un mensaje secreto de prueba para el laboratorio de criptografía!
Seleccione el algoritmo (DES, 3DES, AES-256): AES-256
Ingrese una clave de 32 bytes para AES-256: Esta_es_una_clave_AES_de_32_by

Mensaje cifrado (base64): HEjWQEdwxB+uGsmPzdDdZjwWFyZ+MIxCnvLfa4F5OSH3FezL53tSMLY84f4jDeKLx3p/9+ljPF+dyIkrqT7EVgTx0Lj75fMOL6hmeZzwoww=
(.venv) (base)
```

Figura 12: Output para Clave más Larga, AES-256.

2.4. Implementa el cifrado y descifrado en modo CBC

En este apartado se implemento una función para cada algoritmo de cifrado y descifrado (DES, AES-256, y 3DES) utilizando el modo CBC para todos los algoritmos.

El **modo CBC (Cipher Block Chaining)** es una técnica de cifrado de bloques que introduce dependencia entre los bloques de datos, logrando que el resultado cifrado de cada bloque influya en el siguiente. En este modo, cada bloque de texto plano se combina con el bloque de texto cifrado anterior mediante una operación XOR antes de ser cifrado. Este proceso encadena los bloques, haciendo que la modificación de un bloque afecte a todos los bloques siguientes en el cifrado. Esta interdependencia incrementa la seguridad, ya que impide que el mismo texto cifrado se genere para bloques idénticos en distintas partes del mensaje.
Sin embargo, en el primer bloque no hay un bloque cifrado anterior con el que combinarse, lo que introduce la necesidad de un valor inicial especial: el IV (Initialization Vector).

Por otro lado, el **IV (Vector de Inicialización)** es un valor aleatorio que se utiliza para iniciar el proceso de cifrado en el modo CBC. Para el primer bloque, el IV se combina con el primer bloque de texto plano, reemplazando la función del bloque cifrado anterior. Este valor aleatorio asegura que los primeros bloques de datos cifrados sean únicos cada vez que se cifre el mismo mensaje con la misma clave, lo cual previene que un atacante pueda identificar patrones en los datos cifrados. Por lo tanto, el IV es esencial para proteger la privacidad y evitar ataques de repetición en datos cifrados.

Llevando lo anterior a código, obtenemos lo siguiente:

crypto_cbc.py > main

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

# Función para ajustar la clave a la longitud necesaria
def adjust_key(key, key_size):
    if len(key) < key_size:
        key += get_random_bytes(key_size - len(key))
    elif len(key) > key_size:
        key = key[:key_size]
    return key

# Función para cifrar con DES en modo CBC
def encrypt_des(key, iv, plaintext):
    cipher = DES.new(key, DES.MODE_CBC, iv)
    padded_text = pad(plaintext, DES.block_size)
    encrypted_data = cipher.encrypt(padded_text)
    return base64.b64encode(encrypted_data).decode()

# Función para descifrar con DES en modo CBC
def decrypt_des(key, iv, ciphertext):
    cipher = DES.new(key, DES.MODE_CBC, iv)
    decrypted_data = cipher.decrypt(base64.b64decode(ciphertext))
    return unpad(decrypted_data, DES.block_size).decode()

# Función para cifrar con AES-256 en modo CBC
def encrypt_aes(key, iv, plaintext):
    cipher = AES.new(key, AES.MODE_CBC, iv)
    padded_text = pad(plaintext, AES.block_size)
    encrypted_data = cipher.encrypt(padded_text)
    return base64.b64encode(encrypted_data).decode()

# Función para descifrar con AES-256 en modo CBC
def decrypt_aes(key, iv, ciphertext):
    cipher = AES.new(key, AES.MODE_CBC, iv)
    decrypted_data = cipher.decrypt(base64.b64decode(ciphertext))
    return unpad(decrypted_data, AES.block_size).decode()

# Función para cifrar con 3DES en modo CBC
def encrypt_3des(key, iv, plaintext):
    cipher = DES3.new(key, DES3.MODE_CBC, iv)
    padded_text = pad(plaintext, DES3.block_size)
    encrypted_data = cipher.encrypt(padded_text)
    return base64.b64encode(encrypted_data).decode()

# Función para descifrar con 3DES en modo CBC
def decrypt_3des(key, iv, ciphertext):
    cipher = DES3.new(key, DES3.MODE_CBC, iv)
    decrypted_data = cipher.decrypt(base64.b64decode(ciphertext))
    return unpad(decrypted_data, DES3.block_size).decode()

```

Figura 13: Primera parte del código con modo CBC.

Las funciónes para el **algoritmo DES (Data Encryption Standard)** son las siguientes:

- `encrypt_des`: Esta función cifra el texto plano utilizando DES en modo CBC. Primero, ajusta el texto a un múltiplo del tamaño de bloque de DES mediante la función `pad`. Luego, cifra el texto utilizando la clave y el IV (Vector de Inicialización) proporcionados. El resultado cifrado se codifica en base64 para facilitar su almacenamiento o transmisión.

- `decrypt_des`: Esta función descifra el texto cifrado utilizando DES en modo CBC. El texto cifrado en base64 se decodifica y luego se descifra usando la misma clave y IV que se utilizaron para el cifrado. Finalmente, se elimina el relleno añadido en el proceso de cifrado mediante la función `unpad`, devolviendo el texto original.

Por otro lado, las funciones del **algoritmo 3DES (Triple DES)** son las siguientes:

- `encrypt_3des`: Esta función cifra el texto plano utilizando 3DES en modo CBC. Al igual que con los otros algoritmos, el texto se ajusta con relleno para cumplir con el tamaño de bloque de 3DES. Luego, se cifra con la clave y el IV proporcionados, y el resultado se codifica en base64.

- `decrypt_3des`:Esta función descifra el texto cifrado utilizando 3DES en modo CBC. Primero, se decodifica el texto cifrado en base64, luego se descifra con la misma clave y IV. Finalmente, se elimina el relleno con la función `unpad`, devolviendo el texto original.

Finalmente, las funciones para el **algoritmo Advanced Encryption Standard (AES-256)** son las siguientes:

- `encrypt_aes`: Esta función cifra el texto plano utilizando AES en modo CBC. Al igual que en el caso de DES, primero se aplica el relleno al texto para ajustarlo al tamaño de bloque de AES. Después, el texto se cifra con la clave y el IV proporcionados. El resultado cifrado se codifica en base64 para su fácil manejo.

- `decrypt_aes`: Esta función descifra el texto cifrado utilizando AES en modo CBC. El texto cifrado en base64 se decodifica y luego se descifra utilizando la clave y el IV. Finalmente, se elimina el relleno agregado durante el cifrado con la función `unpad`, devolviendo el texto original.

Cabe destacar que se sigue utilizando la función `adjust_key` del apartado anterior, que tiene como objetivo asegurar que la clave proporcionada por el usuario tenga la longitud correcta para el algoritmo de cifrado seleccionado.

```python
# Función principal que maneja el cifrado y descifrado
def main():
    algorithm = input("Seleccione el algoritmo (DES, 3DES, AES-256): ")
    plaintext = input("Ingrese el mensaje a cifrar: ").encode()
    key_size = 8 if algorithm == 'DES' else (24 if algorithm == '3DES' else 32)
    iv_size = 8 if algorithm in ['DES', '3DES'] else 16

    # Obtener y ajustar clave
    key = input(f"Ingrese una clave de hasta {key_size} bytes para {algorithm}: ").encode()
    key = adjust_key(key, key_size)

    # Obtener y validar IV
    iv = input(f"Ingrese un IV de {iv_size} bytes: ").encode()  # Cambiar de hexadecimal a texto
    if len(iv) != iv_size:
        print(f"El IV debe ser de {iv_size} bytes.")
        return

    # Cifrar y descifrar según el algoritmo
    if algorithm == 'DES':
        encrypted_text = encrypt_des(key, iv, plaintext)
        decrypted_text = decrypt_des(key, iv, encrypted_text)
    elif algorithm == '3DES':
        encrypted_text = encrypt_3des(key, iv, plaintext)
        decrypted_text = decrypt_3des(key, iv, encrypted_text)
    elif algorithm == 'AES-256':
        encrypted_text = encrypt_aes(key, iv, plaintext)
        decrypted_text = decrypt_aes(key, iv, encrypted_text)
    else:
        print("Algoritmo no reconocido.")
        return

    # Mostrar resultados
    print(f"\nMensaje cifrado (base64): {encrypted_text}")
    print(f"Mensaje descifrado: {decrypted_text}")



# Ejecución del programa
if __name__ == "__main__":
    main()

```

Figura 14: Segunda parte del código con modo CBC.

La figura 14 muestra la implementación de la **función main** que permite al usuario seleccionar un algoritmo de cifrado (DES, 3DES o AES-256), ingresar un mensaje, una clave y un IV (Vector de Inicialización) y luego cifrar y descifrar el mensaje utilizando el algoritmo seleccionado en modo CBC.

Una vez explicado el código modificado, se procedera con la ejecución del primer **algoritmo DES.**

```
(.venv) (base)
# lordsamedi @ iSamedi in ~/Desktop/myFolder/Universidad/8vo/lab4 .venv [14:53:21]
$ python3 crypto_cbc.py
Seleccione el algoritmo (DES, 3DES, AES-256): DES
Ingrese el mensaje a cifrar: Este es un mensaje secreto de prueba para el laboratorio de criptografía!
Ingrese una clave de hasta 8 bytes para DES: 12345678
Ingrese un IV de 8 bytes: 26S7eBSu

Mensaje cifrado (base64): qzbgrpZbS8xxFisIp9FkVEb4GefTceC7oSQhYAupzoSo6POVKXq02Dy6gGiC2tR+Hxmltb/U167s1SQWWjWzS8q25ws+AzcDE+F5XOnnGlg=
Mensaje descifrado: Este es un mensaje secreto de prueba para el laboratorio de criptografía!
(.venv) (base)
```

Figura 15: Output algoritmo DES modo CBC.

Tal como se aprecia en la figura 15, como input se dieron los siguientes valores:

- **Mensaje a cifrar**:Este es un mensaje secreto de prueba para el laboratorio de criptografía!
- **Clave de 8 bytes**: *12345678*.
- **IV de 8 bytes**: *26S7eBSu*.

Como resultado se obtuvo el siguiente mensaje cifrado en base64: "zbgrpZbS8xxFisIp9FkVEb4GefTceC7oSQhYAupzoSo6POVKXq02Dy6gGiC2tR+Hxmltb/U167s1SQWWjWzS8q25ws+AzcDE+F5XOnnGlg="; Y al momento de descifrar se obtuvo: *Este es un mensaje secreto de prueba para el laboratorio de criptografía!*, siendo igual al mensaje a cifrar que inicialmente se le dio, logrando un exitoso cifrado y descifrado en modo CBC con el algoritmo DES.

Al ejecutar el segundo algoritmo (3DES), se reutilizo el mensaje a cifrar junto con el vector de inicialización, ya que, en ambos casos su tamaño es de 8 bytes; Utilizando la siguiente clave de 26 bytes:

- **TripleDESde24bytesExactos!**

```
# lordsamedi @ iSamedi in ~/Desktop/myFolder/Universidad/8vo/lab4 .venv [14:53:44]
$ python3 crypto_cbc.py
Seleccione el algoritmo (DES, 3DES, AES-256): 3DES
Ingrese el mensaje a cifrar: Este es un mensaje secreto de prueba para el laboratorio de criptografía!
Ingrese una clave de hasta 24 bytes para 3DES: TripleDESde24bytesExactos!
Ingrese un IV de 8 bytes: 26S7eBSu

Mensaje cifrado (base64): Xwaoh50E0Goworp93n/olCALS7q8STB0zawRhWj/4LT7xlckMrrF+/5+DRV/ZGrwjvTJmY/UtfRMeCVdwNvgpuM9OU9lvjydqHyCPSpgvTs=
Mensaje descifrado: Este es un mensaje secreto de prueba para el laboratorio de criptografía!
(.venv) (base)
```

Figura 16: Output algoritmo 3DES modo CBC.

El output arroja el siguiente mensaje cifrado en base64: "Xwaoh50E0Goworp93n/olCALS7q8STB0zawRhWj/4LT7xlckMrrF+/5+DRV/ZGrwjvTJmY/UtfRMeCVdwNvgpuM9OU9lvjydqHyCPSpgvTs="; Que al descifrarlo se obtiene exitosamente el mensaje ingresado inicialmente.

Finalmente, ejecutando el código utilizando el algoritmo AES-256 con los siguientes parámetros de entrada:

- **Mensaje a cifrar**: Este es un mensaje secreto de prueba para el laboratorio de criptografía!
- **Clave de 34 bytes**: `Esta_es_una_clave_AES_de_32_bytes!`.
- **IV de 16 bytes**: *26S7eBSuDYedBQUM*.

```
(.venv) (base)
# lordsamedi @ iSamedi in ~/Desktop/myFolder/Universidad/8vo/lab4 .venv [15:16:31]
$ python3 crypto_cbc.py
Seleccione el algoritmo (DES, 3DES, AES-256): AES-256
Ingrese el mensaje a cifrar: Este es un mensaje secreto de prueba para el laboratorio de criptografía!
Ingrese una clave de hasta 32 bytes para AES-256: Esta_es_una_clave_AES_de_32_bytes!
Ingrese un IV de 16 bytes: 26S7eBSuDYedBQUM

Mensaje cifrado (base64): F/JfWDXcNnserBjcUACK6qlqIuBhPl3xWcB7cUapyGqFlTfughfNXIhw11V82vszZ+10fdWTME5bealhKuJQ9uLn7Yn0w1MT0wW/cRAIXrc=
Mensaje descifrado: Este es un mensaje secreto de prueba para el laboratorio de criptografía!
(.venv) (base)
```

Figura 17: Output algoritmo AES-256 modo CBC.

Igual que en los casos anteriores, el mensaje descifrado es identico al mensaje a cifrar ingresado en la entrada de la consola, por lo tanto se puede decir que la funcionalibidad de las funciones de cifrado y descifrado en modo CBC funcionan correctamente. Pero nos faltaría realizar una comparación para concluir exitosamente.

2.5. Compara los resultados con un servicio de cifrado online

Para poder comparar los resultados del código del apartado anterior, se utilizo la siguiente página web:

- https://www.davidquinones.dev/es/tools/encode-decode

La página web de David Quińones cuenta con herramientas de encriptación y desencriptación Online, pero no cuenta con el algoritmo DES, en su ausencia trabaja con **DES-EDE** (*Triple DES*). La gran diferencia entre estos dos es que DES-EDE utilia dos llaves a diferencia de DES que utiliza 1, por lo tanto ***DES-EDE-CBC utiliza un tamaño necesario de 16 bytes en su clave***.

Con el fin de utilizar la página antes mencionadas, se realizara un cambio en el programa, especificamente en el algoritmo DES.

```
14  # Función para cifrar con DES-EDE-CBC (Triple DES con dos claves)
15  def encrypt_des_ede(key, iv, plaintext):
16      cipher = DES3.new(key, DES3.MODE_CBC, iv)
17      padded_text = pad(plaintext, DES3.block_size)
18      encrypted_data = cipher.encrypt(padded_text)
19      return base64.b64encode(encrypted_data).decode()
20
21  # Función para descifrar con DES-EDE-CBC (Triple DES con dos claves)
22  def decrypt_des_ede(key, iv, ciphertext):
23      cipher = DES3.new(key, DES3.MODE_CBC, iv)
24      decrypted_data = cipher.decrypt(base64.b64decode(ciphertext))
25      return unpad(decrypted_data, DES3.block_size).decode()
```

Figura 18: Cambios en el código (1).

Se definieron dos funciones para manejar el cifrado y descifrado utilizando **Triple DES (DES-EDE-CBC)** en modo CBC, con dos claves de 16 bytes:

- **Función** `encrypt_des_ede`: Se encarga de cifrar un mensaje utilizando el algoritmo DES-EDE (Triple DES) con el modo de operación CBC y un vector de inicialización (IV) proporcionado por el usuario.
- **Función** `decrypt_des_ede`: Su función es descifrar un mensaje previamente cifrado con **Triple DES en modo CBC**

```
58      # Configuración de tamaño de clave e IV
59      if algorithm == 'DES-EDE':  # DES-EDE-CBC (dos claves, 16 bytes)
60          key_size = 16
61          iv_size = 8
62      elif algorithm == '3DES':  # DES-EDE3-CBC (tres claves, 24 bytes)
63          key_size = 24
64          iv_size = 8
65      elif algorithm == 'AES-256':
66          key_size = 32
67          iv_size = 16
68      else:
69          print("Algoritmo no reconocido.")
70          return
```

Figura 19: Cambios en el código (2).

Tal como se mencionó anteriormente, al ser un Triple DES con dos llaves, posee un tamaño de clave de 16 bytes, ya que, corresponde al uso de dos claves de 8 bytes para el cifrado. Mientras que el **IV (Vector de Inicialización)** para el modo CBC en DES es de 8 bytes. Este es un tamaño estándar para DES y 3DES (incluyendo el modo EDE).

Una vez explicado y modificado todo lo necesario, procedemos a realizar el cifrado por la página online y compararlos con los resultados arrojados por el código Python.

Comenzando con el **algoritmo DES-EDE**, se utilizará los siguientes input en la pagina web:

- **Mensaje**: Este es un mensaje secreto de prueba para el laboratorio de criptografía!
- **Clave de 16 bytes**: *1234567891234567*.

Figura 20: Resultado DES-EDE-CBC online.

Como se observa en los resultados arrojados por la herramienta online, el mensaje cifrado es: "kSa6CKZHX9rN9hJzvTHWs6ZR8b7kk0TuKTVTDyH7eMkEXpLvr45aTpPa6o9UDPo3mjSH391gmt1i+XkDsUj9zhUveqNC1/Ig0/2A7fTOy0w=". Por otro lado, en la pestaña de 'Más Información' se observa un *vector*, que hace referencía al ***vector de inicialización o IV***, el cual es ***26S7eBSu*** con un tamaño de 8 bytes.

Una vez obtenido el vector, se dirigio al código Python, ingresando los mismos inputs y clave, junto con el IV generado por la página online.

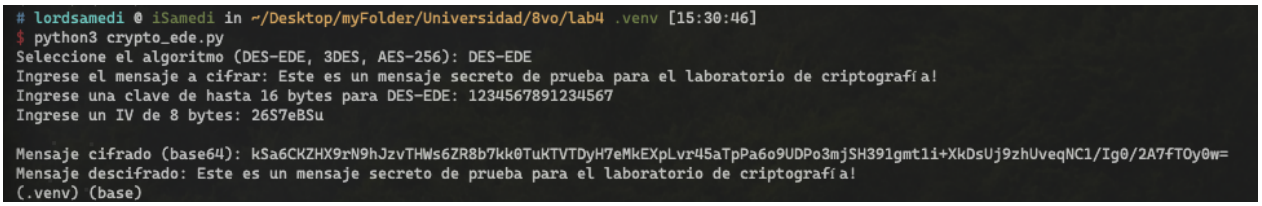

Figura 21: Resultado DES-EDE-CBC código Python.

El mensaje cifrado a tráves del código es: "kSa6CKZHX9rN9hJzvTHWs6ZR8b7kk0TuKTVTDyH7eMkEXpLvr45aTpPa6o9UDPo3mjSH391gmt1i+XkDsUj9zhUveqNC1/Ig0/2A7fTOy0w=". El mensaje cifrado por Python es identico al mensaje generado Online utilizando el mismo vector de inicialización, concluyendo que el funcionamiento de Descifrado y Cifrado por DES-EDE, funciona correctamente.

En segundo lugar, para el **algoritmo 3DES** o **DES-EDE3-CBC** (*nombre que posee en la página web*), se ingresaron los siguientes valores de entrada a la página web:

- **Mensaje**: Este es un mensaje secreto de prueba para el laboratorio de criptografía!
- **Clave**: TripleDESde24bytesExactos!

Figura 22: Resultado DES-EDE3-CBC online.

El mensaje cifrado es: "Xwaoh50E0Goworp93n/olCALS7q8STB0zawRhWj/4LT7xlckMrrF+/5+DRV/ZGrwjvTJmY/UtfRMeCVdwNvgpuM9OU9lvjydqHyCPSpgvTs=". Igual que en

el caso anterior, se utilizara el vector de inicialización de la página web, ***26S7eBSu***, siendo identico al del caso anterior y con un tamaño de 8 bytes.

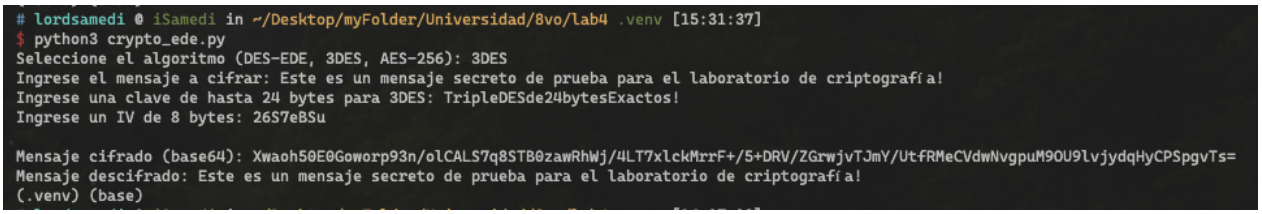

Figura 23: Resultado DES-EDE3-CBC código Python.

Al reutilizar el mensaje, clave y el vector de inicialización dado por la página web, se obtuvo el siguiente mensaje cifrado en base64: "Xwaoh50E0Goworp93n/olCALS7q8STB0zawRhWj/4LT7xlckMrrF+/5+DRV/ZGrwjvTJmY/UtfRMeCVdwNvgpuM9OU9lvjydqHyCPSpgvTs=". Ambos mensajes son identicos al utilizar igual vector de inicialización, por lo tanto, se concluye que el código funciona correctamente.

Finalmente, el **algoritmo AES-256**, utiliza el mismo mensaje que en los dos casos anteriores, la clave en este caso posee un tamaño de 32 bytes siendo el siguiente:

- `Esta_es_una_clave_AES_de_32_bytes!`.

Figura 24: Resultado DES-EDE3-CBC online.

El mensaje cifrado a tráves de la página web es el siguiente: "F/JfWDXcNnserBjcUACK6qlqIuBhPl3xWcB7cUapyGqFlTfughfNXIhw11V82vszZ+10fdWTME5bealhKuJQ9uLn7Yn0w1MT0wW/cRAIXrc=", con un vector de 16 bytes "***26S7eBSuDYedBQUM***".

```
(.venv) (base)
# lordsamedi @ iSamedi in ~/Desktop/myFolder/Universidad/8vo/lab4 .venv [16:35:20]
$ python3 crypto_ede.py
Seleccione el algoritmo (DES-EDE, 3DES, AES-256): AES-256
Ingrese el mensaje a cifrar: Este es un mensaje secreto de prueba para el laboratorio de criptografía!
Ingrese una clave de hasta 32 bytes para AES-256: Esta_es_una_clave_AES_de_32_bytes!
Ingrese un IV de 16 bytes: 26S7eBSuDYedBQUM

Mensaje cifrado (base64): F/JfWDXcNnserBjcUACK6qlqIuBhPl3xWcB7cUapyGqFlTfughfNXIhw11V82vszZ+10fdWTME5bealhKuJQ9uLn7Yn0w1MT0wW/cRAIXrc=
Mensaje descifrado: Este es un mensaje secreto de prueba para el laboratorio de criptografía!
(.venv) (base)
```

Figura 25: Resultado DES-EDE3-CBC código Python.

El mensaje cifrado en base64 a tráves del código Python es el siguiente: "F/JfWDXcNnserBjcUACK6qlqIuBhPl3xWcB7cUapyGqFlTfughfNXIhw11V82vszZ+10fdWTME5bealhKuJQ9uLn7Yn0w1MT0wW/cRAIXrc=", siendo identico al mensaje cifrado por la página online, corroborando el excelente funcionamiento del código a la hora de cifrar y descifrar para los tres algoritmos (DES, DES-EDE, 3DES y AES-256).

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

El **cifrado simétrico** juega un papel esencial en la seguridad de la información en la vida diaria debido a su rapidez y simplicidad. En este tipo de cifrado, **una misma clave** *es utilizada tanto para cifrar como para descifrar los datos*, lo cual resulta en un método eficaz para proteger la información en muchos contextos comunes.

Un ejemplo significativo de su aplicación se encuentra en las **transacciones bancarias y financieras**. Para asegurar que la información personal y financiera de los usuarios esté protegida, las aplicaciones bancarias emplean cifrado simétrico para datos como el PIN, las credenciales de usuario y los detalles de las transacciones. Este tipo de cifrado permite que la transmisión de información sea rápida y segura, protegiendo al usuario contra posibles interceptaciones. En este contexto, el **algoritmo AES (Advanced Encryption Standard)** es **ampliamente utilizado** debido a su *balance entre seguridad y velocidad, siendo ideal para proteger pagos y transferencias electrónicas en tiempo real.*

Otra aplicación esencial del cifrado simétrico es la s**eguridad de las conexiones de red**, especialmente en ***Internet***. El protocolo HTTPS emplea cifrado simétrico para proteger la comunicación entre un navegador web y el servidor al que se conecta, garantizando la privacidad de la información transmitida. De manera similar, las **redes privadas virtuales (VPNs) utilizan cifrado simétrico** para mantener la seguridad de la conexión entre un usuario y una red privada, haciendo que la comunicación sea segura sin perder la fluidez de la conexión, un aspecto crucial para aplicaciones en tiempo real.

Conclusiones y comentarios

El laboratorio brindó una experiencia completa sobre el funcionamiento y la importancia del cifrado simétrico para proteger información en la vida diaria. Se implementaron algoritmos como DES, 3DES y AES en modo CBC, aprendiendo a cifrar y descifrar datos de manera segura usando claves y vectores de inicialización (IV).

Estos conceptos son fundamentales para garantizar la privacidad y seguridad de datos sensibles frente a accesos no autorizados. El laboratorio también permitió entender por qué el cifrado simétrico es preferido en situaciones que requieren velocidad y eficiencia, como en la transmisión de grandes volúmenes de datos o aplicaciones en tiempo real.

En resumen, el laboratorio ayudó a comprender cómo el cifrado simétrico es un componente vital en la seguridad de nuestra información diaria, desde la banca en línea hasta la mensajería privada y el almacenamiento seguro. Estos ejercicios permitieron aplicar sus conocimientos de criptografía de una forma práctica, adquiriendo habilidades para manejar técnicas de cifrado en situaciones reales y aumentando su comprensión de las herramientas necesarias para proteger datos en entornos digitales.