

Informe Laboratorio 4

Sección 1

Alumno Sebastian Silva
e-mail: sebastian.silva_b@mail.udp.cl

Noviembre de 2024

Índice

1. Descripción de actividades	2
2. Desarrollo de actividades según criterio de rúbrica	3
2.1. Investiga y documenta los tamaños de clave e IV	3
2.2. Solicita datos de entrada desde la terminal	4
2.3. Valida y ajusta la clave según el algoritmo	4
2.4. Implementa el cifrado y descifrado en modo CBC	5
2.5. Compara los resultados con un servicio de cifrado online	6
2.6. Describe la aplicabilidad del cifrado simétrico en la vida real	7

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 esta experiencia, es fundamental saber los tamaños en bytes de la clave y el vector de inicialización para DES, AES-256 y 3DES, estos se especifican a continuación:

- DES: Requiere una clave de 8 bytes (64 bits) y 8 bytes para el vector de inicialización (IV).
- 3DES: Requiere una clave de 24 bytes (192 bits, que en realidad son tres claves de 8 bytes) y 8 bytes para el vector de inicialización (IV).
- AES-256: Requiere una clave de 32 bytes (256 bits) y 16 bytes para el vector de inicialización (IV).

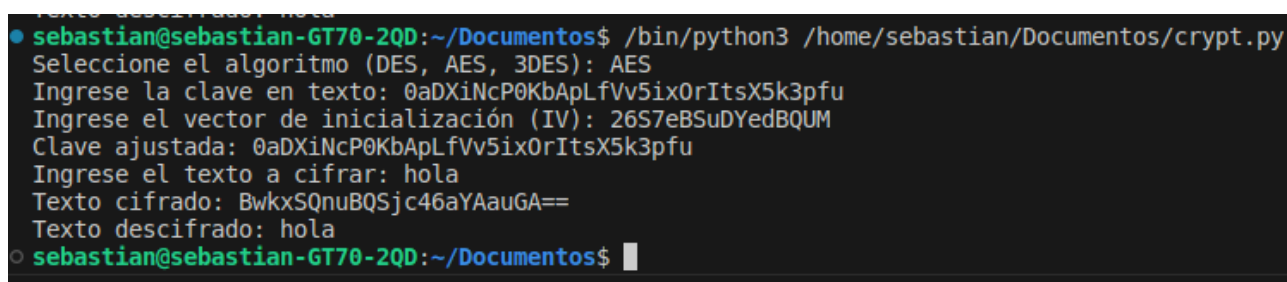
Una vez aclarado los tamaños de las claves y del vector de inicialización se debe entender las principales diferencias entre estos:

- DES: Es el más antiguo y menos seguro debido a su tamaño de clave limitado de 64 bits, de los cuales solo 56 bits son efectivos, ya que se tiene que los otros 8 se usan para paridad. Tiene una velocidad rápida, pero con una baja seguridad.
- 3DES: Este algoritmo aplica DES 3 veces, incrementando de este modo su seguridad en comparación con DES. No obstante, es más lento debido a las tres rondas de cifrado, y aunque mejora la seguridad, sigue siendo menos eficiente que AES.
- AES-256: Es el más moderno, seguro y eficiente de los 3, soportando tamaños de clave de hasta 256 bits y es ampliamente utilizado en la protección de datos confidenciales debido a su combinación de alta seguridad y velocidad.

Cabe recalcar que esta información se recolecto buscando información de 'National Institute of Standards and Technology (NIST)', se busco específicamente de los siguientes links:
<https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-67r1.pdf>
<https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf>

2.2. Solicita datos de entrada desde la terminal

Para el desarrollo del código, se implementaron los algoritmos DES, AES-256 y 3DES, en donde se proba el funcionamiento de AES-256. Para esto se solicitarán ciertos datos mediante la terminal, los cuales son: Tipo de algoritmo, clave en texto, vector de inicialización en texto y el texto a cifrar. De este modo se obtiene lo siguiente:

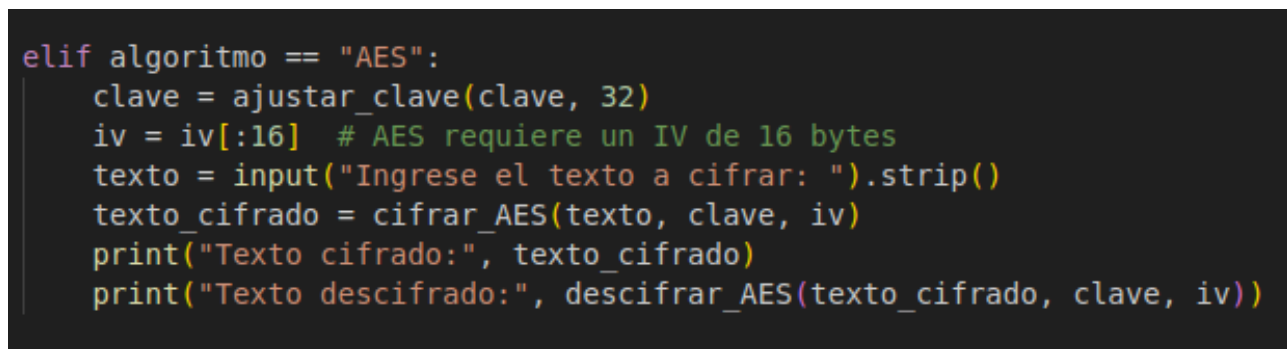


```
sebastian@sebastian-GT70-2QD:~/Documentos$ /bin/python3 /home/sebastian/Documentos/crypt.py
Seleccione el algoritmo (DES, AES, 3DES): AES
Ingrese la clave en texto: 0aDXiNcP0KbApLfVv5ix0rItsX5k3pfu
Ingrese el vector de inicialización (IV): 26S7eBSuDYedBQUM
Clave ajustada: 0aDXiNcP0KbApLfVv5ix0rItsX5k3pfu
Ingrese el texto a cifrar: hola
Texto cifrado: BwkxSQnuBQSjc46aYAauGA==
Texto descifrado: hola
sebastian@sebastian-GT70-2QD:~/Documentos$
```

Figura 1: Solicitud de datos desde terminal

2.3. Valida y ajusta la clave según el algoritmo

Para este punto se busca validar y ajustar la clave según el algoritmo utilizado, para este caso, se usará AES-256. El código empleado para las funcionalidades solicitadas se puede apreciar a continuación:



```
elif algoritmo == "AES":
    clave = ajustar_clave(clave, 32)
    iv = iv[:16] # AES requiere un IV de 16 bytes
    texto = input("Ingrese el texto a cifrar: ").strip()
    texto_cifrado = cifrar_AES(texto, clave, iv)
    print("Texto cifrado:", texto_cifrado)
    print("Texto descifrado:", descifrar_AES(texto_cifrado, clave, iv))
```

Figura 2: Visualización del llamado a la función de ajuste de la clave

Como se logra ver, se tiene el llamado a la función encargada de ajustar la clave, de modo que pueda asegurar que la longitud de la clave se cumpla con los requisitos del algoritmo. La función revisa la longitud y, si es necesario, completa o trunca la clave ingresada.

```
# Función para ajustar la clave según el tamaño requerido
def ajustar_clave(clave, tamaño):
    clave_bytes = clave.encode("utf-8")
    if len(clave_bytes) < tamaño:
        clave_bytes += get_random_bytes(tamaño - len(clave_bytes))
    elif len(clave_bytes) > tamaño:
        clave_bytes = clave_bytes[:tamaño]
    print(f"Clave ajustada: {clave_bytes.decode('utf-8', errors='ignore')}")
    return clave_bytes
```

Figura 3: Visualización de ajuste de la clave según el algoritmo

Por ultimo, en la figura 3, se muestra el ajuste de la clave, es decir, el proceso que valida y modifica la clave. Si se tiene una clave más corta que el tamaño requerido, se rellenan los bytes faltantes de manera controlada, asegurando de esta forma que el cifrado funcione correctamente al estandarizar la longitud de la clave para el algoritmo utilizado.

2.4. Implementa el cifrado y descifrado en modo CBC

Para el siguiente punto se debe implementar tanto en el cifrado como en el descifrado en modo CBC. Esto se puede apreciar a continuación mediante los recuadros en rojo:

```
# Funciones de cifrado y descifrado para AES-256
def cifrar_AES(texto, clave, iv):
    cipher = AES.new(clave, AES.MODE_CBC, iv)
    texto_cifrado = cipher.encrypt(pad(texto.encode("utf-8"), AES.block_size))
    return base64.b64encode(texto_cifrado).decode("utf-8")

def descifrar_AES(texto_cifrado, clave, iv):
    cipher = AES.new(clave, AES.MODE_CBC, iv)
    texto_cifrado_bytes = base64.b64decode(texto_cifrado.encode("utf-8"))
    texto_descifrado = unpad(cipher.decrypt(texto_cifrado_bytes), AES.block_size)
    return texto_descifrado.decode("utf-8")
```

Figura 4: Visualización de la implementación del cifrado y descifrado en modo CBC

Como se logra ver, se muestra el flujo de implementación de las funciones de cifrado y descifrado en CBC, donde se puede también, la inicialización del cifrado usando la clave y el IV proporcionados, la aplicación de padding para que el texto se ajuste al tamaño de bloque del algoritmo y el cifrado del texto plano y su conversión a un formato de salida.

2.5. Compara los resultados con un servicio de cifrado online

Se utilizo la pagina <https://www.devglan.com/online-tools/aes-encryption-decryption> para cifrar el texto en AES-256 y comparar con lo obtenido en el codigo, de este modo se obtuvo lo siguiente:

AES Encryption

Enter Plain Text to Encrypt

hola

Select Cipher Mode of Encryption ?

CBC

Select Padding ?

PKCS5Padding

Enter IV (Optional) ?

26S7eBSuDYedBQUM

Key Size in Bits ?

256

Enter Secret Key ?

0aDXiNcP0KbApLfVv5ixOrltsX5k3pfu

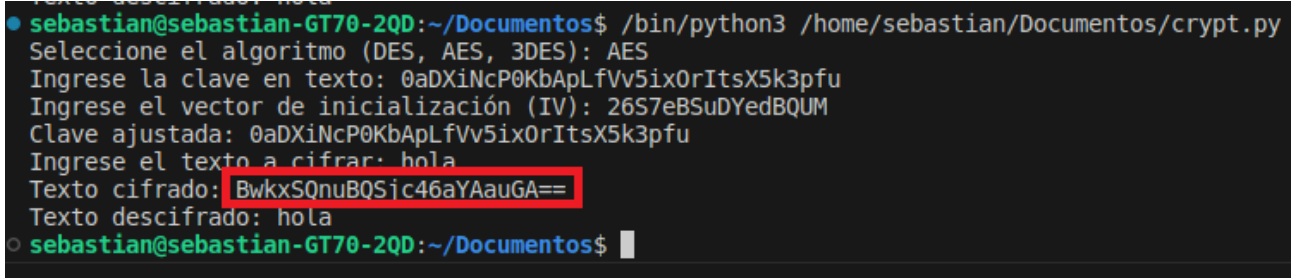
Output Text Format ☒ Base64 ☐ Hex

Encrypt

AES Encrypted Output

BwxxSQnuBQSjc46aYAauGA==

Figura 5: Visualización de los resultados obtenidos en un cifrado online

A terminal window with a dark background and green text. The prompt is 'sebastian@sebastian-GT70-2QD:~/Documentos\$'. The user runs '/bin/python3 /home/sebastian/Documentos/crypt.py'. The program prompts for an algorithm (AES), a key (0aDXiNcP0KbApLfVv5ix0rItsX5k3pfu), an IV (26S7eBSuDYedBQUM), and a text to encrypt ('hola'). It then displays the encrypted text 'BwkxSQnuBQSjc46aYAauGA==' which is highlighted with a red rectangle. Finally, it shows the decrypted text 'hola' and returns to the shell prompt.

```
sebastian@sebastian-GT70-2QD:~/Documentos$ /bin/python3 /home/sebastian/Documentos/crypt.py
Seleccione el algoritmo (DES, AES, 3DES): AES
Ingrese la clave en texto: 0aDXiNcP0KbApLfVv5ix0rItsX5k3pfu
Ingrese el vector de inicialización (IV): 26S7eBSuDYedBQUM
Clave ajustada: 0aDXiNcP0KbApLfVv5ix0rItsX5k3pfu
Ingrese el texto a cifrar: hola
Texto cifrado: BwkxSQnuBQSjc46aYAauGA==
Texto descifrado: hola
sebastian@sebastian-GT70-2QD:~/Documentos$
```

Figura 6: Visualización de los resultados obtenidos en el código implementado

Como se puede apreciar en el recuadro rojo, se tiene el mismo texto cifrado, el cual es 'BwkxSQnuBQSjc46aYAauGA=='. De este modo, se demuestra el funcionamiento del código implementado.

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

Un caso de uso práctico para la aplicabilidad del cifrado simétrico en la vida real es la protección de datos confidenciales en bases de datos. Por ejemplo, en sistemas bancarios o médicos, donde se almacenan datos personales sensibles, es esencial mantener la seguridad de la información. Para este contexto, el algoritmo AES-256 es ideal debido a su alto nivel de seguridad y eficiencia de cifrado y descifrado, permitiendo proteger grandes volúmenes de datos sin comprometer el rendimiento.

En el caso que la contraparte sugiere utilizar hashes en lugar de cifrado simétrico, sería importante aclarar que los hashes son métodos de resúmenes de datos y, a diferencia del cifrado, no permiten recuperar la información original. Los hashes son útiles para verificar integridad, pero no para encriptar y descifrar la información. En este caso, explicar la diferencia y la función específica del cifrado simétrico en mantener la confidencialidad ayudaría a justificar por qué AES-256 es la opción adecuada para este tipo de aplicación.

Conclusiones y comentarios

A modo de cierre, se tiene que la implementación de los algoritmos de cifrado simétrico (DES, AES-256 y 3DES) utilizando el modo CBC, con claves y vectores de inicialización fue exitosa. La experiencia permitió entender la importancia de los ajustes en la longitud de las claves y la correcta aplicación del padding para un cifrado eficaz y seguro. Además, se comprobó la precisión del código al comparar los resultados con un servicio de cifrado en línea, validando su funcionamiento.

El análisis también resaltó las aplicaciones del cifrado simétrico en situaciones de la vida real, como la transmisión segura de datos y el almacenamiento de información confidencial, destacando su eficiencia en entornos de tiempo real. Para finalizar, se puede decir que el cifrado simétrico sigue siendo relevante y efectivo para proteger datos sensibles en diversos contextos.