Informe Laboratorio 4

Sección 2

Cristóbal Barra
cristobal.barra1@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

Primero que todo, en la página web de PyCrypto se indica que la librería **pycrypto** se encuentra desactualizada, la misma página hace recomendación del uso de la librería **pycryptodome**, que es una rama de la librería anterior mencionada.

Con respecto a las claves e IVs, sus tamaños dependen de cada algoritmo de cifrado y su forma de trabajar. El tamaño de las claves se basa en el nivel de seguridad requerido de la época en la cual fue creado el algoritmo de cifrado, asimismo de las limitaciones de éstos mismos. Tanto DES como 3DES utilizan claves de 64 bits (8 bytes) de los cuales los últimos 8 bits son reservados para la verificación de paridad. Por su parte, AES-256 es un algoritmo de cifrado moderno que utiliza claves de 256 bits (32 bytes), lo que proporciona una seguridad mucho mayor. Esta información se puede encontrar dentro de Geeks for Geeks, donde se realiza una comparación entre AES y DES, y se señalan los tamaños de las claves que se utilizan para cifrar.

Los vectores de inicialización son un factor clave utilizado en el modo de operación CBC, y se utilizan para realizar la operación XOR con el primer bloque del método (información obtenida de la Academia Europea de Certificación de Tecnologías de la Información), de este modo creando una cadena de textos operaciones XOR con los bloques cifrados anteriores. Por lo tanto, el tamaño de los vectores de inicialización dependen del tamaño de los bloques con los cuales se operan, 8 bytes para DES y 16 bytes para AES-256, esta información también se puede encontrar en la comparativa entre estos algoritmos señalada anteriormente.

2.2. Solicita datos de entrada desde la terminal

Para que los algoritmos funcionen correctamente, se necesitan tres factores: llave, vector de inicialización (IV) y texto a cifrar. Estos deben ser solicitados por la terminal e ingresados por el usuario. El input para cada algoritmo es diferente, pues en 3DES se piden tres llaves para el cifrado, y en AES-256 se pide una llave de 32 bytes (4 veces mayor a DES) y un IV de 16 bytes (2 veces mayor a DES).
Cifrado DES:

```
key = adjust_key(input('Llave (8 bytes): '))
iv = input('IV (8 bytes): ')
plaintext = input('Texto: ')
```

Figura 1: Input cifrado DES.

Cifrado 3DES:
Cabe destacar, que las tres llaves sean distintas entre sí. La llave 1 (K1) se utiliza para cifrar, la llave 2 (K2) para descifrar y la llave 3 (K3) para cifrar nuevamente. Si K3 es igual a cualquiera de las otras dos llaves, el algoritmo sería equivalente a un 2DES, en cualquier otro caso (quitando que todas las llaves sean distintas) el algoritmo se reduce a un simple DES.

```
print('Llaves (8 bytes)')
key1 = adjust_key(input("  Llave 1: "))
key2 = adjust_key(input("  Llave 2: "))
key3 = adjust_key(input("  Llave 3: "))

iv = input("IV (8 bytes): ")
plaintext = input("Texto: ")
```

Figura 2: Input cifrado 3DES.

Cifrado AES-256:
El input de AES-256 es similar a DES, solo que aumenta el tamaño de la llave y el IV.

```
key = adjust_key(input('Llave (32 bytes): '))
iv = input("IV (16 bytes): ")
plaintext = input("Texto: ")
```

Figura 3: Input cifrado AES-256.

2.3. Valida y ajusta la clave según el algoritmo

En las instrucciones se especifica que si la llave es más corta que lo requerido, ésta sea rellenada con bytes aleatorios, y si es más larga entonces se trunque a la cantidad ed bytes que se requieren. La función vista en la siguiente figura realiza justamente eso, con el método get_random_bytes().

```
def adjust_key(key):
    key = key.encode('utf-8')
    print(f'  Llave ingresada (hex): {binascii.hexlify(key).decode('utf-8')}')
    if len(key) > 8:
        key = key[:8]
    elif len(key) < 8:
        key += get_random_bytes(8 - len(key))
    print(f'  Llave ajustada (hex): {binascii.hexlify(key).decode('utf-8')}')
    return key
```

```
def adjust_key(key):
    key = key.encode('utf-8')
    print(f'  Llave ingresada (hex): {binascii.hexlify(key).decode('utf-8')}')
    if len(key) > 32:
        key = key[:32]
    elif len(key) < 32:
        key += get_random_bytes(32 - len(key))
    print(f'  Llave ajustada (hex): {binascii.hexlify(key).decode('utf-8')}')
    return key
```

Figura 4: Ajuste de llave para 8 y 32 bytes.

Para DES y 3DES se utiliza la función de la izquierda, para AES-256 se usa la función de la derecha, la función también imprime la llave ingresada en el input y la llave ajustada, ambas en formato hexadecimal. En el caso de que la llave sea más corta a los bytes requeridos, se rellenará con bytes aleatorios, tal como se ve en la figura 5.

```
Llave (8 bytes): hola
  Llave ingresada (hex): 686f6c61
  Llave ajustada (hex): 686f6c61b19edafb
```

Figura 5: Llave ingresada más corta de lo requerido.

Si la llave ingresada es más larga que lo indicado, entonces simplemente se trunca hasta llegar a los 8 bytes. Se puede apreciar en la figura 6.

```
Llave (8 bytes): hola que tal todo
  Llave ingresada (hex): 686f6c61207175652074616c20746f646f
  Llave ajustada (hex): 686f6c6120717565
```

Figura 6: Llave ingresada más larga de lo requerido.

Las dos figuras anteriores son para el caso de DES y 3DES, para el caso de AES-256 funciona de igual manera solo que el largo requerido es de 32 bytes.

2.4. Implementa el cifrado y descifrado en modo CBC

El estándar para trabajar DES es de bloques con 8 bytes y AES son bloques de 32 bytes, dentro del código se utilizaron esos mismos valores para operar. Las funciones reciben tanto la llave, como el IV y el texto a cifrar o descifrar. Como usuarios ingresamos la llave, IV y texto en formato ASCII, pero la función necesita trabajar en bits, por lo tanto codifica a UTF-8 los inputs para realizar dicha labor.

Como trabaja con bloques preestablecidos, si es que el texto a cifrar o descifrar no tiene un largo múltiplo de estas cifras entonces se utiliza el padding. El padding que utiliza DES, 3DES y AES por defecto es PKCS#7.

Para implementar el modo de operación CBC hace falta una sola línea, por ejemplo, para DES solo hizo falta escribir **cipher = DES.new(key, DES.MODE_CBC, iv)**, línea la cual crea una instancia que funciona como cifrador o descifrador según su requerimiento, y recibe como parámetros la llave, el modo de operación y el IV en caso de necesitarse. Las funciones de cifrado y descifrado vistas en las figuras 7, 8 y 9, funcionan de manera muy similar, con ciertos cambios en cada una para un correcto funcionamiento.

```python
def des_encrypt(key, iv, plaintext):
    iv = iv.encode('utf-8')
    plaintext = plaintext.encode('utf-8')
    if len(plaintext) % BLOCK_SIZE == 0:
        padded_text = plaintext
    else:
        padded_text = pad(plaintext, BLOCK_SIZE)
    cipher = DES.new(key, DES.MODE_CBC, iv)
    ciphertext = cipher.encrypt(padded_text)
    return binascii.hexlify(ciphertext).decode('utf-8'), key, iv

def des_decrypt(key, iv, ciphertext):
    ciphertext = binascii.unhexlify(ciphertext)
    cipher = DES.new(key, DES.MODE_CBC, iv)
    decrypted_padded_text = cipher.decrypt(ciphertext)
    padding_length = decrypted_padded_text[-1]
    decrypted_text = unpad(decrypted_padded_text, BLOCK_SIZE)
    return decrypted_text.decode('utf-8')
```

Figura 7: Funciones de encriptado y desencriptado DES.

```python
def des3_encrypt(key, iv, plaintext):
    iv = iv.encode('utf-8')
    plaintext = plaintext.encode('utf-8')
    padded_text = pad(plaintext, BLOCK_SIZE)
    cipher = DES3.new(key, DES3.MODE_CBC, iv)
    ciphertext = cipher.encrypt(padded_text)
    return binascii.hexlify(ciphertext).decode('utf-8')

def des3_decrypt(key, iv, ciphertext):
    iv = iv.encode('utf-8')
    ciphertext = binascii.unhexlify(ciphertext)
    cipher = DES3.new(key, DES3.MODE_CBC, iv)
    decrypted_padded_text = cipher.decrypt(ciphertext)
    decrypted_text = unpad(decrypted_padded_text, BLOCK_SIZE).decode('utf-8')
    return decrypted_text
```

Figura 8: Funciones de encriptado y desencriptado 3DES.

```python
def aes_encrypt(key, iv, plaintext):
    iv = iv.encode('utf-8')
    plaintext = plaintext.encode('utf-8')
    padded_text = pad(plaintext, BLOCK_SIZE)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    ciphertext = cipher.encrypt(padded_text)
    return binascii.hexlify(ciphertext).decode('utf-8')

def aes_decrypt(key, iv, ciphertext):
    iv = iv.encode('utf-8')
    ciphertext = binascii.unhexlify(ciphertext)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    decrypted_padded_text = cipher.decrypt(ciphertext)
    decrypted_text = unpad(decrypted_padded_text, BLOCK_SIZE).decode('utf-8')
    return decrypted_text
```

Figura 9: Funciones de encriptado y desencriptado AES-256.

En resumen, las funciones necesitan codificar los parámetros recibidos, aplicar padding en caso de necesitarse para los bloques y emplear el cifrado o descifrado. Para el caso del descifrado, hay que quitar el padding luego de desencriptar el texto y decodificarlo para mostrarlo en formato ASCII.

Como ejemplo de la implementación de los algoritmos, se utilizarán llaves con el largo requerido por cada cifrador y el mismo texto a cifrar, solo un ejemplo por implementación, para no entorpecer lo que se quiere demostrar.

Output DES:

```
Llave (8 bytes): holahola
  Llave ingresada (hex): 686f6c61686f6c61
  Llave ajustada (hex): 686f6c61686f6c61
IV (8 bytes): holahola
Texto: hola que tal

==========================

Texto cifrado (hex): 83fdb66869b76874c19d7b324475bf94

==========================

Texto descifrado: hola que tal
```

Figura 10: Ejemplo de output DES.

Output 3DES:

```
Llaves (8 bytes)
  Llave 1: holahola
    Llave ingresada (hex): 686f6c61686f6c61
    Llave ajustada (hex): 686f6c61686f6c61
  Llave 2: hola que
    Llave ingresada (hex): 686f6c6120717565
    Llave ajustada (hex): 686f6c6120717565
  Llave 3: holawasa
    Llave ingresada (hex): 686f6c6177617361
    Llave ajustada (hex): 686f6c6177617361
IV (8 bytes para 3DES): holahola
Texto: hola que tal

==========================

Texto cifrado con 3DES (hex): 36266cd63bb2819ef5fbf470745910dd

==========================

Texto descifrado con 3DES: hola que tal
```

Figura 11: Ejemplo de output 3DES.

Output AES-256:

```
Llave (32 bytes): holaholaholaholaholaholaholahola
  Llave ingresada (hex): 686f6c61686f6c61686f6c61686f6c61686f6c61686f6c61686f6c61686f6c61
  Llave ajustada (hex): 686f6c61686f6c61686f6c61686f6c61686f6c61686f6c61686f6c61686f6c61
IV (16 bytes): holaholaholahola
Texto: hola que tal

==========================

Texto cifrado con AES-256 (hex): cd819da556a2df0bd9b0472275e91127

==========================

Texto descifrado con AES-256: hola que tal
```

Figura 12: Ejemplo de output AES-256.

2.5. Compara los resultados con un servicio de cifrado online

Si utilizamos el ejemplo de la figura 10 y su mismo input dentro de la página web para cifrar con DES, `https://anycript.com/crypto/des`. Ingresamos los siguientes datos de la figura 13, y se observa que el output es exactamente el mismo, o sea **83fdb66869b76874c19d7b324475b** por lo tanto para claves del largo requerido el algoritmo trabaja de igual manera al implementado en python.

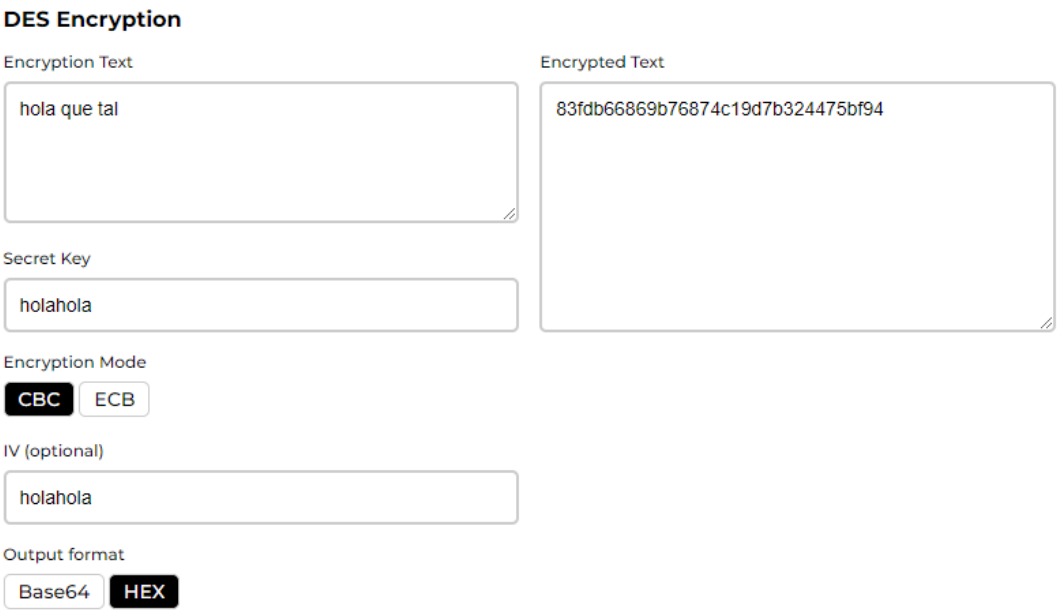

Figura 13: Output con clave de 8 bytes.

Para llaves más cortas que el requerido de 8 bytes, la implementación en python rellena con bytes aleatorios por cada vez que se ejecute el código. En cambio, el cifrado de la página siempre muestra el mismo output de la figura 14, esto quiere decir que rellena las llaves de una forma específica y no de manera aleatorio como el código en python.

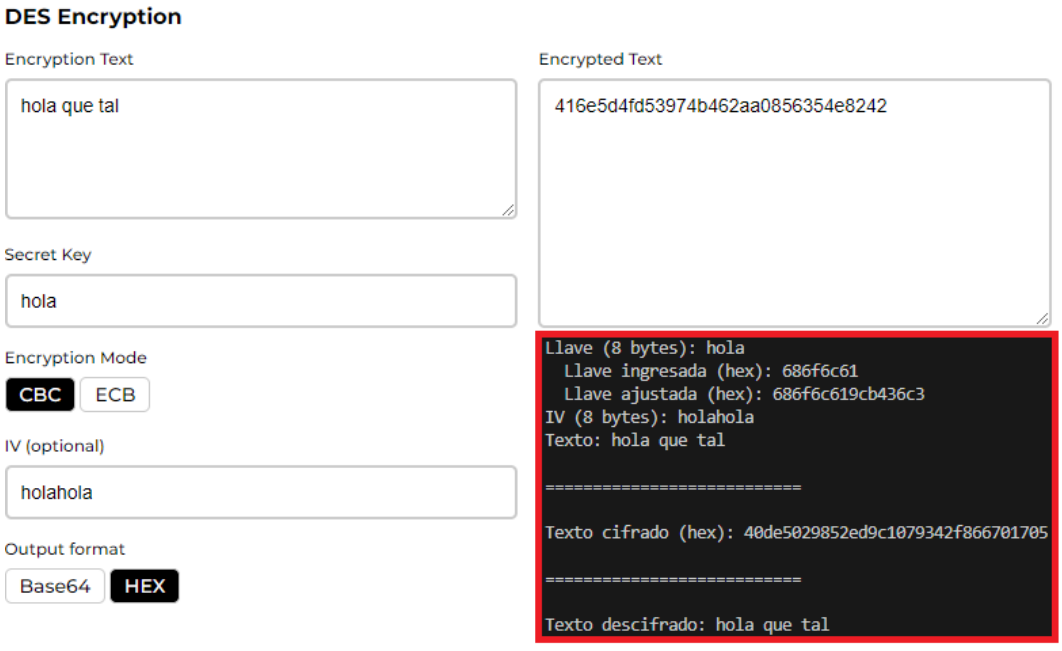

Figura 14: Output con clave de 4 bytes.

Para claves más largas del requerido, ambos algoritmos truncan la llave ingresada a 8 bytes y cifran de la misma manera. Cabe destacar que el texto "hola que tal" es de 12 bytes, por lo que se debe hacer padding a los 4 bytes faltantes, el código en python utiliza el padding PKCS#7, como en las figuras 13 y 15 se observan el mismo output, entonces ambas implementaciones utilizan el mismo tipo de padding.

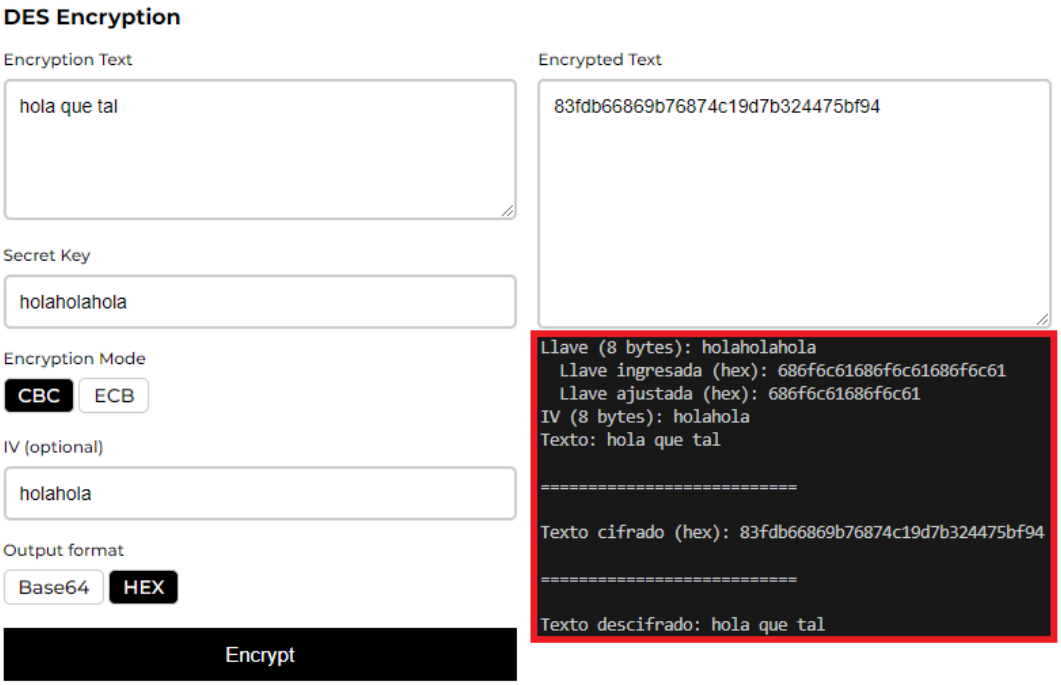

Figura 15: Output con clave de 12 bytes.

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

Si se habla de aplicabilidad del cifrado simétrico en la vida real, hay que tomar en cuenta que hay niveles de seguridad dentro de los tipos de cifrado, ya se entendió dentro de este laboratorio que AES-256 es mucho más seguro que DES o 3DES, por lo que tratará la aplicabilidad de los cifrados simétricos más seguros. Estos métodos permiten un alto nivel de confidencialidad, además de que se puede cifrar y descifrar rápidamente, como en el caso de los algoritmos recién vistos, que su ejecución demora milisegundos.

El cifrado simétrico se puede utilizar para el traspaso de archivos de manera confidencial, de manera que los archivos sean visibles solo por las peronas que interactúan en el intercambio. Como estos archivos pertenecen a empresas, esta información puede ser de carácter sensible, por lo que cifrar la información es crucial para mantener seguros los datos.

En el caso de que mi contraparte me pidiera implementar hashes en vez de cifrado simétrico, habría que explicarle el hasheo de los datos no permite que la información hasheada sea

reversible, por lo que la información quedaría ilegible para ambos, sin manera de recuperar los datos. Por lo general el hash se utiliza para verificar la integridad de los datos, algo que no viene al caso dentro de mantener confidencialidad del traspaso de datos y archivos. En resumen no sería una buena implementación el uso de hash en vez de cifrado simétrico.

Conclusiones y comentarios

Para terminar con este laboratorio, el cifrado simétrico es una herramienta imprescindible dentro de la seguridad de la información debido a la eficiencia en la protección de los datos. Con la implementación de los algoritmos de cifrado simétrico se obtuvo una idea de como es que operan estos algoritmos por dentro y como trabajan con los datos que se le ingresan, tanto con la codificación como con el relleno de los bloques. AES-256 resulta ser el algoritmo más seguro dentro de los tres vistos, con su robustez en las claves y modo interno de operar, permiten resguardar la información de mejor manera.

La comparación con el hash, aunque corta, también permitió entender en qué escenarion se deben utilizar cada uno. Además, se pudo comprender el uso del modo de operación CBC, el cuál trabaja con cadenas de bloques los cuales se cifran con su bloque cifrado anterior. Factores como estos influyen en la robustez del cifrado, así como también la complejidad del vector de inicialización.