Informe Laboratorio 4

Sección 4

Renato Oscar Benjamin Contreras Carvajal
e-mail: renato.contreras@mail.udp.cl

Noviembre de 2024

Índice

1. Descripción de actividades **2**

2. Desarrollo de actividades según criterio de rúbrica **3**
2.1. Investigue y documente los tamaños de clave e IV 3
2.2. Solicita datos de entrada desde la terminal 3
2.3. Valida y ajusta la clave según el algoritmo 5
2.4. Implementación del cifrado y descifrado en modo CBC 6
2.5. Comparación de resultados con un servicio de cifrado en línea 7
2.6. Describe la aplicabilidad del cifrado simétrico en la vida real 9

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. Investigue y documente los tamaños de clave e IV

Para determinar los tamaños requeridos en bytes para el funcionamiento de los algoritmos DES, AES-256 y 3DES, se consultaron las especificaciones disponibles en PyCryptodome, utilizando las siguientes fuentes:

- `https://www.pycryptodome.org/src/cipher/des#module-Crypto.Cipher.DES` para DES.
- `https://www.pycryptodome.org/src/cipher/aes#aes` para AES-256.
- `https://www.pycryptodome.org/src/cipher/des3` para 3DES.

Los tamaños de clave e IV requeridos para cada algoritmo son:

- **DES:** Tamaño de clave de 8 bytes (64 bits), aunque solo 56 bits son efectivos debido al uso de bits de paridad. Tamaño del IV: 8 bytes (64 bits).
- **AES-256:** Tamaño de clave de 32 bytes (256 bits). Tamaño del IV: 16 bytes (128 bits).
- **3DES:** Tamaño de clave de 24 bytes (192 bits, utilizando tres claves de 64 bits con 56 bits efectivos cada una). Tamaño del IV: 8 bytes (64 bits).

En cuanto a las diferencias principales entre los algoritmos, DES utiliza una clave relativamente corta, lo que lo hace vulnerable a ataques de fuerza bruta debido a su limitado nivel de seguridad. Por otro lado, AES-256 ofrece un nivel de seguridad significativamente superior gracias a su clave de 256 bits y es ampliamente reconocido como el estándar moderno para cifrado. Finalmente, 3DES mejora la seguridad de DES aplicando tres iteraciones del cifrado, pero a costa de una mayor lentitud en comparación con AES, además de tener una longitud de clave efectiva más baja que la de AES-256.

2.2. Solicita datos de entrada desde la terminal

Para la segunda parte del laboratorio, se desarrolló un código en Python que permite al usuario proporcionar los datos necesarios para realizar el cifrado o descifrado. A continuación, se describen los pasos implementados:

El programa comienza ofreciéndole dos opciones principales: elegir entre un encriptador o un desencriptador. A continuación, solicita al usuario seleccionar uno de los tres algoritmos disponibles: DES, AES-256 o 3DES. Según la elección realizada, se establecen los tamaños requeridos para la clave (key) y el vector de inicialización (IV). Posteriormente, para cada algoritmo, el programa valida que la clave y el IV proporcionados tengan el tamaño correcto. En caso de que no cumplan con este requisito, se muestra un mensaje de error y se solicita nuevamente la entrada correspondiente, asegurando así que los algoritmos puedan operar adecuadamente en el modo CBC. Finalmente, el programa captura el texto a cifrar, junto con la clave y el IV, y los muestra en consola como confirmación.

El siguiente código Python implementa lo antes mencionado:

```
from Crypto.Random import get_random_bytes

def get_validated_input(prompt, expected_size, description):
    while True:
        user_input = input(f"{prompt} ({description}): ").encode()
        if len(user_input) == expected_size:
            return user_input
        print(f"Error: La entrada debe tener exactamente {expected_size} bytes.")

def main():
    print("Bienvenido al encriptador y desencriptador")
    print("1. Encriptador")
    print("2. Desencriptador")

    while True:
        choice = input("Seleccione una opcion (1 o 2): ")
        if choice in ['1', '2']:
            break
        print("Error: Debe seleccionar 1 o 2.")

    print("\nQue algoritmo quieres usar? Se ocupa el modo CBC en cada uno.")
    print("1. DES")
    print("2. AES-256")
    print("3. 3DES")

    while True:
        algorithm_choice = input("Seleccione un algoritmo (1, 2 o 3): ")
        if algorithm_choice in ['1', '2', '3']:
            break
        print("Error: Debe seleccionar 1, 2 o 3.")

    # Determinar requisitos en funcion del algoritmo seleccionado
    if algorithm_choice == '1':  # DES
        key_size = 8
        iv_size = 8
        algorithm_name = "DES"
    elif algorithm_choice == '2':  # AES-256
        key_size = 32
        iv_size = 16
```

```
        algorithm_name = "AES-256"
    else:  # 3DES
        key_size = 24
        iv_size = 8
        algorithm_name = "3DES"

    print(f"\n--- {algorithm_name} ---")
    plaintext = input("Ingrese texto a cifrar: ").encode()

    # Validar la clave y el IV
    key = get_validated_input(
        "Ingrese la clave",
        key_size,
        f"{key_size} bytes requeridos para {algorithm_name}"
    )
    iv = get_validated_input(
        "Ingrese el Vector de Inicializacion (IV)",
        iv_size,
        f"{iv_size} bytes requeridos para {algorithm_name}"
    )

    print("\nDatos capturados correctamente.")
    print(f"Texto a procesar: {plaintext.decode()}")
    print(f"Clave: {key}")
    print(f"IV: {iv}")

if __name__ == "__main__":
    main()
```

Listing 1: Código para solicitar y validar datos de entrada

2.3. Valida y ajusta la clave según el algoritmo

Si bien la sección anterior tenía en cuenta que se ingresaran los campos y se configuró para que fueran exactos, en este apartado se nos indica cómo y qué hacer en caso de que la clave sea menor o mayor a la requerida. Para cumplir con este requisito, implementamos la siguiente función, que ajusta la clave ingresada según el tamaño necesario para cada algoritmo:

```
def get_input_with_adjustment(prompt, target_size, description):
    user_input = input(f"{prompt} ({description}): ").encode()

    # Ajustar la clave
    if len(user_input) < target_size:
        # Completar clave con get_random_bytes
        padding = get_random_bytes(target_size - len(user_input))
        adjusted_key = user_input + padding
    elif len(user_input) > target_size:
        # Truncar clave
        adjusted_key = user_input[:target_size]
    else:
```

```
        # La clave ya tiene el tamano correcto
        adjusted_key = user_input

    # Imprimir clave final ajustada
    print(f"Clave ajustada: {adjusted_key}")
    return adjusted_key
```

Listing 2: Código para solicitar y validar datos de entrada

2.4. Implementación del cifrado y descifrado en modo CBC

Para las funciones de cifrado y descifrado, nos apoyamos directamente en la librería `pycryptodome`: `https://www.pycryptodome.org/`. Las siguientes funciones son las ocupadas para realizar las operaciones de cifrado y descifrado con en el modo CBC:

```
def encrypt_des(plaintext, key, iv):
    cipher = DES.new(key, DES.MODE_CBC, iv)
    padded_text = plaintext + b" " * (8 - len(plaintext) % 8)
    return cipher.encrypt(padded_text)

def decrypt_des(ciphertext, key, iv):
    cipher = DES.new(key, DES.MODE_CBC, iv)
    decrypted_text = cipher.decrypt(ciphertext)
    return decrypted_text.rstrip()

def encrypt_aes(plaintext, key, iv):
    cipher = AES.new(key, AES.MODE_CBC, iv)
    padded_text = plaintext + b" " * (16 - len(plaintext) % 16)
    return cipher.encrypt(padded_text)

def decrypt_aes(ciphertext, key, iv):
    cipher = AES.new(key, AES.MODE_CBC, iv)
    decrypted_text = cipher.decrypt(ciphertext)
    return decrypted_text.rstrip()

def encrypt_3des(plaintext, key, iv):
    cipher = DES3.new(key, DES3.MODE_CBC, iv)
    padded_text = plaintext + b" " * (8 - len(plaintext) % 8)
    return cipher.encrypt(padded_text)

def decrypt_3des(ciphertext, key, iv):
    cipher = DES3.new(key, DES3.MODE_CBC, iv)
    decrypted_text = cipher.decrypt(ciphertext)
    return decrypted_text.rstrip()
```

Listing 3: Código para solicitar y validar datos de entrada

Con todo esto realizado, obtenemos el código final, el cual incluye funciones adicionales para ingresar los datos correctamente y asegurar el funcionamiento del programa. El archivo completo se encuentra disponible en el repositorio bajo el nombre `Lab4-criptografia.py`.

```
rena@rena-cc:~/Downloads$ /usr/bin/python3 /home/rena/Downloads/Lab4-criptografia.py
Bienvenido al encriptador y desencriptador

Que algoritmo quieres usar? Se ocupa el modo CBC en cada uno.
1. DES
2. AES-256
3. 3DES
Seleccione un algoritmo (1, 2 o 3): 3

--- 3DES ---
Ingrese texto a cifrar: hola
Ingrese la clave (24 bytes requeridos para 3DES): 123456789101112131415167
Clave ajustada: 123456789101112131415167
Ingrese el Vector de Inicializacion (IV) (8 bytes requeridos para 3DES): chaochao

Datos capturados correctamente.
Texto a procesar: hola
Clave final utilizada: 123456789101112131415167
IV final utilizado: chaochao

Texto cifrado (hexadecimal): b9889da84d196154
Texto descifrado: hola
rena@rena-cc:~/Downloads$
```

Figura 1: Salida del código completo.

Para replicar el resultado, utilice los siguientes datos de entrada:

- **Algoritmo:** 3
- **Texto a cifrar:** `hola`
- **Clave:** `123456789101112131415167`
- **Vector de Inicialización (IV):** `chaochao`

Con estos valores de entrada, se obtiene:

- **Texto cifrado (hexadecimal):** `b9889da84d196154`
- **Texto descifrado:** `hola`

2.5. Comparación de resultados con un servicio de cifrado en línea

Para verificar los resultados del cifrado 3DES, utilizamos los mismos valores en la página web `https://www.devglan.com/online-tools/triple-des-encrypt-decrypt`. A continuación, se muestra una captura de pantalla de los resultados obtenidos en esta herramienta en línea.

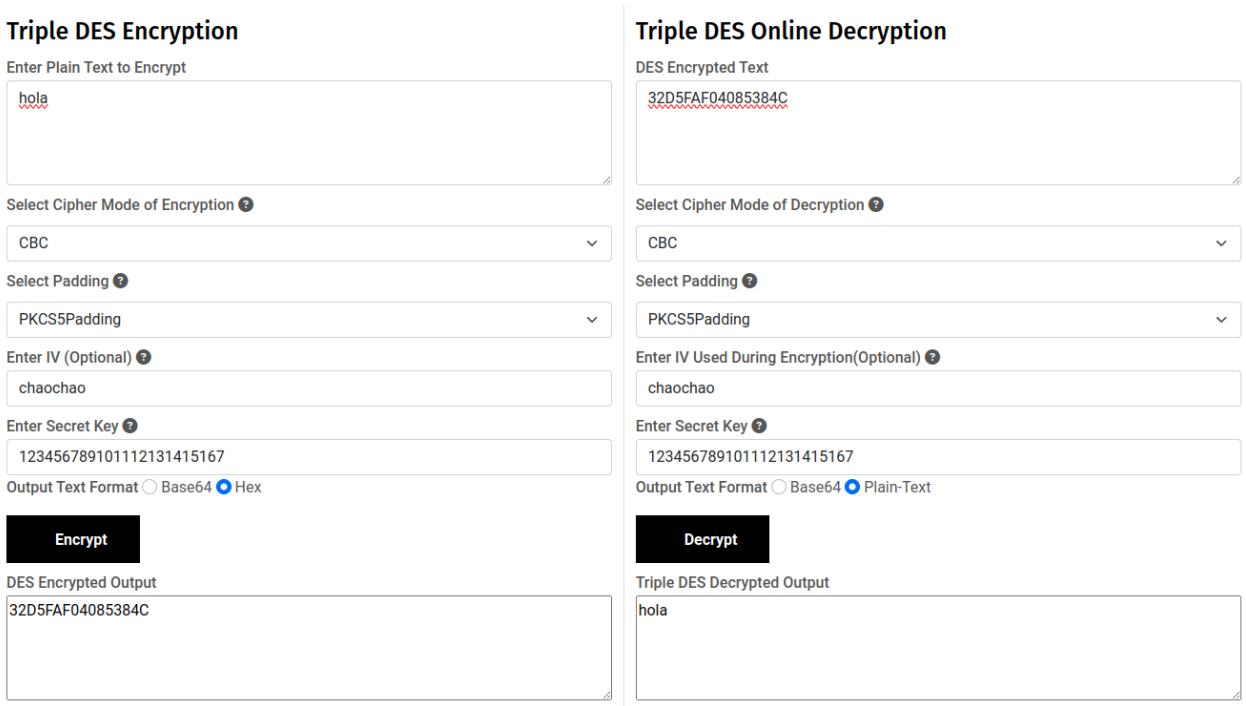

Figura 2: Cifrado y descifrado con 3DES en una herramienta en línea.

Al comparar los resultados, se observa una diferencia entre el texto cifrado generado por el código en Python y el obtenido en la página web. En el código, el resultado del cifrado es `b9889da84d196154`, mientras que en la herramienta en línea es `32D5FAF04085384C`.

La diferencia en los resultados entre el código y la web podría deberse a cómo cada implementación maneja internamente el proceso de encriptación de Triple DES. En el código Python, DES3.new se utiliza para inicializar el cifrado Triple DES con una clave de 24 bytes, aplicándola directamente sin subdividirla explícitamente en tres claves (K1, K2 y K3). Por otro lado, la herramienta en línea podría implementar una lógica que divide la clave en tres segmentos de 8 bytes o que ocupa la clave de manera diferente durante las fases de encriptación, desencriptación y re-encriptación de Triple DES. Por lo que estas variaciones en el uso de la clave podrían ser las que generan diferencias en el texto cifrado final, aun usando el mismo algoritmo y modo CBC.

También consideré la posibilidad de que la diferencia estuviera en el padding. En el código Python, los bloques que no alcanzan el tamaño requerido se rellenan con espacios en blanco, mientras que la web usa PKCS5, un esquema de padding estandarizado que añade bytes específicos en función de la longitud de los datos. Sin embargo, en este caso, el tamaño de la entrada es adecuado (es múltiplo de 8) para evitar padding adicional, por lo que esta opción no explica del todo la diferencia observada en los resultados.

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

En situaciones reales, como la comunicación entre bancos o grandes empresas que manejan datos sensibles y se necesita acceder a la información original, usaría el cifrado simétrico ya que sirve para proteger la confidencialidad de la información. Específicamente ocuparía el algoritmo AES-256 ya que este es uno de los estándares de cifrado más fuertes y usados en la industria por su alta resistencia a ataques y eficacia con grandes datos. AES-256 tiene un buen equilibrio entre seguridad y eficiencia y hasta fue adoptado como estándar de cifrado por el gobierno de los Estados Unidos [1]. Esto demuestra su aplicabilidad en un ámbito empresarial que necesita seguridad y rapidez.

Si la recomendación de usar AES-256 no fuera bien recibida y la contraparte sugiriera usar funciones hash en lugar de cifrado simétrico, sería importante aclarar la diferencia entre ambos enfoques. Las funciones hash, como SHA-256, son herramientas que garantizan la integridad de los datos, ya que generan un resumen único y no reversible de la información. Sin embargo, el hash no permite recuperar el contenido original, por lo que no seria recomendable cuando la confidencialidad es prioritaria y se necesita acceder nuevamente a la información original.

Los hashes son útiles para detectar cambios o manipulación de los datos, como en la verificación de integridad de archivos o la protección de contraseñas, pero no ofrecen una capa de confidencialidad. En este caso, reafirmaría que el cifrado simétrico con AES-256 es la elección adecuada para comunicaciones seguras, ya que permite tanto el cifrado como el descifrado, protegiendo los datos en tránsito y garantizando que solo las partes autorizadas puedan acceder a la información.

Conclusiones y comentarios

En el laboratorio se exploraron las características y diferencias teóricas entre los algoritmos de cifrado DES, AES-256 y 3DES, considerando factores como la longitud de clave y su aplicabilidad en contextos de seguridad. Por lo que según lo aprendido en la teoría y la práctica, AES-256 es el más valorado por su robustez y eficiencia. En comparación, DES presenta limitaciones debido a su clave corta, lo que lo hace vulnerable a ataques. Y aunque 3DES mejora la seguridad de DES, su eficiencia es menor que la de AES-256, lo que lo hace menos adecuado para aplicaciones modernas que requieren rapidez y alta seguridad.

Por otro lado, el análisis del caso me permitió comprender mejor la aplicabilidad de los algoritmos de cifrado simétrico frente a las funciones hash en distintos contextos. Aunque ambos métodos contribuyen a la protección de datos, es evidente que el cifrado simétrico es preferible cuando se requiere tanto confidencialidad como acceso a la información original. Como expliqué anteriormente, las funciones hash, al ser unidireccionales e irreversibles, son útiles para verificar la integridad de los datos o proteger contraseñas, pero no proporcionan confidencialidad ni permiten recuperar el contenido original. Esto me ayudó a entender la importancia de seleccionar el método adecuado según las necesidades específicas de seguridad, integridad y acceso.

Referencias

[1] Prey Project. *Tipos de cifrado: simétrico o asimétrico, RSA o AES*. Consultado en noviembre de 2024. 2024. URL: https://preyproject.com/es/blog/tipos-de-cifrado-simetrico-o-asimetrico-rsa-o-aes.