Informe Laboratorio 4

Sección 4

Alumno Felipe Fernandez
e-mail: felipe.fernandez1@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

El algoritmo DES (Data Encryption Standard), un cifrador simétrico, utiliza una clave de 56 bits efectiva, dentro de un bloque de 64 bits que incluye 8 bits de paridad, dejando una longitud funcional de 56 bits para el cifrado. La estructura de DES divide el mensaje en bloques de 64 bits, utilizando una clave común tanto para cifrar como para descifrar. En modos como CBC (Cipher Block Chaining) o CFB (Cipher Feedback), DES también emplea un vector de inicialización (IV) de 64 bits para evitar que patrones repetitivos en el texto claro generen patrones idénticos en el texto cifrado, lo que añade un nivel adicional de seguridad frente a ataques de análisis estadístico.

AES-256, por otro lado, pertenece a la familia de cifrados simétricos de bloque avanzados y utiliza una clave de 256 bits, que proporciona una mayor resistencia a ataques de fuerza bruta en comparación con DES y 3DES. AES cifra bloques de 128 bits y requiere un IV del mismo tamaño (128 bits) en modos operativos como CBC. Este mayor tamaño de clave y bloque permite a AES-256 ofrecer una protección criptográfica robusta y eficiente, ampliamente utilizada en sistemas de alta seguridad y regulada por estándares internacionales como FIPS-197.

3DES (Triple DES) es una extensión del algoritmo DES que aplica el proceso de cifrado en tres etapas consecutivas, utilizando tres claves de 56 bits para alcanzar una longitud efectiva de 168 bits. Esta metodología, aunque más segura que el DES original, es significativamente más lenta y menos eficiente, y aún utiliza un IV de 64 bits. Por esta razón, 3DES ha sido progresivamente reemplazado por AES, especialmente en aplicaciones que requieren tanto alta seguridad como eficiencia operativa.

2.2. Solicita datos de entrada desde la terminal

Para llevar a cabo esta actividad, se desarrolló un código en Python que recopila los datos proporcionados a través de la terminal. Para ello, se empleó la librería **PyCryptodome**, que facilita la implementación de diversos algoritmos criptográficos, permitiendo la ejecución de tareas de cifrado y desencriptado de información. El código resultante es el siguiente:

```
def obtener_clave_y_iv(algoritmo):
    #Se solicita al usuario la clave de cifrado
    clave = input("Ingresa la clave de cifrado (texto plano): ").encode('utf-8')

    #Se solicita el IV y se valida su longitud
    while True:
        iv_hex = input(f"Ingresa el vector de iniciación (IV) en formato hexadecimal p
        #Se valida que el IV (que está en hexadecimal) posea una cantidad de dígitos p
        try:
            iv = bytes.fromhex(iv_hex)
            #Se valida que posea una longitud adecuada al algoritmo
            if not es_iv_valido(iv, algoritmo):
                longitud_requerida = obtener_longitud_iv(algoritmo)
                print(f"Error: El IV para {algoritmo} debe tener una longitud de {long
                continue
            break
        #Se lanza una excepción si la longitud del IV no es par
        except ValueError:
            print("Error: El IV proporcionado no es un formato hexadecimal válido.")
    #Se solicita al usuario el texto a cifrar
    texto = input("Ingresa el texto a cifrar: ").encode('utf-8')

    #Se retorna la clave, el IV y el texto a cifrar
    return clave, iv, texto
```

El código presentado recibe la clave de cifrado en texto plano (ASCII); en este laboratorio se ha elegido la clave **llavetest** por su sencillez. El código también recibe el vector de inicialización (IV) en formato hexadecimal, validando que cumpla con la longitud requerida para el algoritmo de cifrado seleccionado. En el caso del algoritmo DES, el IV utilizado es **AABBCCDDEEFF0011**, que posee la longitud de 8 bytes correspondiente tanto para DES como para 3DES. Para el algoritmo AES-256, el IV elegido es **AABBCCDDEEFF0011AABBCCDDEEFF0011**, con una longitud de 16 bytes adecuada para dicho algoritmo. Finalmente, el código solicita el mensaje a cifrar en texto plano, para el cual se ha elegido **texto a cifrar** como ejemplo de entrada.

```
Selecciona el algoritmo de cifrado:
1. DES
2. AES-256
3. 3DES
Ingresa el número correspondiente a la opción: 1
Ingresa la clave de cifrado (texto plano): llavetest
Ingresa el vector de iniciación (IV) en formato hexadecimal para DES: AABBCCDDEEFF0011
Ingresa el texto a cifrar: texto a cifrar
```

Figura 1: datos ingresados en la terminal

2.3. Valida y ajusta la clave según el algoritmo

Posteriormente, se incorporaron al código funciones que verifican si el IV cumple con la longitud requerida para el algoritmo de cifrado seleccionado. Adicionalmente, se añadió una función que permite agregar bits aleatorios a la clave cuando esta es menor a la longitud mínima exigida por el algoritmo, y otra función para truncarla si supera el límite máximo permitido. Cabe destacar que el código excluye de manera intencional los caracteres ASCII no imprimibles, con el fin de evitar problemas en la actividad 5, que requiere una clave compatible con plataformas de cifrado en línea.

```
def es_iv_valido(iv, algoritmo):
    #Se recupera la longitud de IV requerida por el algoritmo
    longitud_requerida = obtener_longitud_iv(algoritmo)

    #Retorna True si la longitud del IV proporcionada es la del algoritmo
    return len(iv) == longitud_requerida

def obtener_longitud_iv(algoritmo):
    #Se retorna la longitud requerida por el algoritmo
    if algoritmo == "DES" or algoritmo == "3DES":
        return 8  # IV de 8 bytes para DES y 3DES
    elif algoritmo == "AES-256":
        return 16  # IV de 16 bytes para AES-256
    else:
        raise ValueError("Algoritmo no soportado para validación de IV.")

def obtener_bytes_aleatorios_ascii(tamano):
    #Se generan bytes aleatorios imprimibles en ASCII, para propiciar el uso
    #en el sitio web, que sólo acepta ASCII para las claves
    bytes_imprimibles = []

    #Se repite el proceso hasta alcanzar el tamaño requerido por el algoritmo
```

```
    while len(bytes_imprimibles) < tamano:
        #Se genera una cantidad de bytes aleatorios
        random_bytes = get_random_bytes(tamano * 2)  # Generamos más bytes de los nece
        for byte in random_bytes:
            #Solo se añaden los bytes imprimibles (32-126 en ASCII)
            if 33 <= byte <= 126:
                bytes_imprimibles.append(byte)
            if len(bytes_imprimibles) >= tamano:
                break
    #Se retornan los bytes aleatorios
    return bytes(bytes_imprimibles[:tamano])

def ajustar_clave(clave, algoritmo):
    #Se ajusta la clave al tamaño requerido por el algoritmo
    if algoritmo == "DES":
        longitud = 8  #DES usa claves de 8 bytes
    elif algoritmo == "AES-256":
        longitud = 32  #AES-256 usa claves de 32 bytes
    elif algoritmo == "3DES":
        longitud = 24  #3DES usa claves de 24 bytes
    else:
        raise ValueError("Algoritmo no soportado.")

    if len(clave) < longitud:
        #Se completa con bytes aleatorios imprimibles en ASCII
        clave = clave + obtener_bytes_aleatorios_ascii(longitud - len(clave))
    elif len(clave) > longitud:
        #Se trunca la clave
        clave = clave[:longitud]

    return clave
```

```
Selecciona el algoritmo de cifrado:
1. DES
2. AES-256
3. 3DES
Ingresa el número correspondiente a la opción: 1
Ingresa la clave de cifrado (texto plano): llavetest
Ingresa el vector de iniciación (IV) en formato hexadecimal para DES: AABBCCDDEEF
Ingresa el texto a cifrar: texto a cifrar

Clave ajustada (en ASCII): 'llavetes'
```

Figura 2: Respuesta de la terminal con la llave ajustada

Como se observa en la imagen anterior, indicada por la flecha roja, la clave **llavetest** fue truncada a **llavetes** debido a que excedía la longitud máxima permitida para el algoritmo DES.

2.4. Implementa el cifrado y descifrado en modo CBC

Finalmente, se añadió la función que permite cifrar y descifrar el mensaje utilizando el algoritmo previamente seleccionado, así como la opción de elegir el algoritmo de cifrado implementado en la función **def main():**.

```
def cifrar_y_descifrar(clave, iv, texto, algoritmo):
    #Se ajusta la clave al tamaño necesario
    clave_ajustada = ajustar_clave(clave, algoritmo)
    clave_ascii = clave_ajustada.decode('ascii')
    #Se imprime la clave ajustada
    print(f"\nClave ajustada (en ASCII): '{clave_ascii}'")

    if algoritmo == "DES":
        #Se crea el objeto de cifrado DES
        cipher = DES.new(clave_ajustada, DES.MODE_CBC, iv)
        texto_padded = pad(texto, DES.block_size)
        texto_cifrado = cipher.encrypt(texto_padded)
        #Descifrado
        decipher = DES.new(clave_ajustada, DES.MODE_CBC, iv)
        texto_descifrado = unpad(decipher.decrypt(texto_cifrado), DES.block_size)

    elif algoritmo == "AES-256":
        #Se crea el objeto de cifrado AES
        cipher = AES.new(clave_ajustada, AES.MODE_CBC, iv)
        texto_padded = pad(texto, AES.block_size)
        texto_cifrado = cipher.encrypt(texto_padded)
        #Descifrado
        decipher = AES.new(clave_ajustada, AES.MODE_CBC, iv)
        texto_descifrado = unpad(decipher.decrypt(texto_cifrado), AES.block_size)

    elif algoritmo == "3DES":
        #Se crea el objeto de cifrado 3DES
        cipher = DES3.new(clave_ajustada, DES3.MODE_CBC, iv)
        texto_padded = pad(texto, DES3.block_size)
        texto_cifrado = cipher.encrypt(texto_padded)
        #Descifrado
        decipher = DES3.new(clave_ajustada, DES3.MODE_CBC, iv)
        texto_descifrado = unpad(decipher.decrypt(texto_cifrado), DES3.block_size)

    else:
        raise ValueError("Algoritmo no soportado.")

    #Se devuelven los resultados
```

```
    return texto_cifrado, texto_descifrado.decode('utf-8')

def main():
    print("Selecciona el algoritmo de cifrado:")
    print("1. DES")
    print("2. AES-256")
    print("3. 3DES")

    opcion = input("Ingresa el número correspondiente a la opción: ")

    if opcion == "1":
        algoritmo = "DES"
    elif opcion == "2":
        algoritmo = "AES-256"
    elif opcion == "3":
        algoritmo = "3DES"
    else:
        print("Opción no válida.")
        return

    clave, iv, texto = obtener_clave_y_iv(algoritmo)
    texto_cifrado, texto_descifrado = cifrar_y_descifrar(clave, iv, texto, algoritmo)

    #Se muestran los resultados
    print("Texto original:", texto.decode('utf-8'))
    print("Texto cifrado (en hexadecimal):", binascii.hexlify(texto_cifrado).decode('u
    print("Texto descifrado:", texto_descifrado)

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

```
Selecciona el algoritmo de cifrado:
1. DES
2. AES-256
3. 3DES
Ingresa el número correspondiente a la opción: 1
Ingresa la clave de cifrado (texto plano): llavetest
Ingresa el vector de iniciación (IV) en formato hexadecimal para DES: AABBCCDDEEFF0011
Ingresa el texto a cifrar: texto a cifrar

Clave ajustada (en ASCII): 'llavetes'
Texto original: texto a cifrar
Texto cifrado (en hexadecimal): cc418628a33d0bcbbdc9a0287929a208
Texto descifrado: texto a cifrar

Process finished with exit code 0
```

Figura 3: cifrado del codigo utilizando DES

```
1. DES
2. AES-256
3. 3DES
Ingresa el número correspondiente a la opción: 2
Ingresa la clave de cifrado (texto plano): llavetest
Ingresa el vector de iniciación (IV) en formato hexadecimal para AES-256: AABBCCDDEEFF0011AABBCCDDEEFF0011
Ingresa el texto a cifrar: texto a cifrar

Clave ajustada (en ASCII): 'llavetest5|*YHQc#(o`jL%I"/<oV9fz'
Texto original: texto a cifrar
Texto cifrado (en hexadecimal): 8517aeff6c75d2ea045fdca0c5a5a32a
Texto descifrado: texto a cifrar

Process finished with exit code 0
```

Figura 4: cifrado del codigo utilizando AES - 256

```
Selecciona el algoritmo de cifrado:
1. DES
2. AES-256
3. 3DES
Ingresa el número correspondiente a la opción: 3
Ingresa la clave de cifrado (texto plano): llavetest
Ingresa el vector de iniciación (IV) en formato hexadecimal para 3DES: AABBCCDDEEFF0011
Ingresa el texto a cifrar: texto a cifrar

Clave ajustada (en ASCII): 'llavetestqiOrf}5#)if%zu^'
Texto original: texto a cifrar
Texto cifrado (en hexadecimal): 69af1bd245ca45df0f427c2b50bb186c
Texto descifrado: texto a cifrar

Process finished with exit code 0
```

Figura 5: cifrado del codigo utilizando 3DES

En las tres imágenes anteriores se observa que el código cifró el mensaje **texto a cifrar**, generando el resultado **cc418628a33d0bcbbdc9a0287929a208** al aplicar el cifrado DES, **8517aeff6c75d2ea045fdca0c5a5a32a** con el algoritmo AES-256 y **69af1bd245ca45df0f427c2b50bb** utilizando 3DES, como se señala con la flecha roja. Asimismo, se presenta el mensaje descifrado en cada una de las imágenes, indicado con una flecha azul. Cabe mencionar que las claves empleadas para AES-256 y 3DES fueron ajustadas mediante la adición de bytes para cumplir con los requisitos de longitud de cada algoritmo, como se indica mediante flechas verdes.

2.5. Compara los resultados con un servicio de cifrado online

Para verificar el correcto funcionamiento del código, se utilizó la página **DES Decrypt Online** para descifrar el mensaje obtenido tras cifrarlo con el algoritmo DES. El mensaje cifrado se ingresó en el campo designado como **Ingrese la cadena a descifrar con DES**, tal como se muestra en la imagen 6, destacada por un rectángulo azul. Además, se completaron otros campos requeridos, como **Formato del mensaje cifrado**, **Clave**, **IV**, entre otros. Luego, se seleccionó el botón **DES Descifrar**, lo que permitió obtener el mensaje descifrado, mostrado en el rectángulo rojo. Este mensaje coincidió con el texto original utilizado previamente para realizar los cifrados con los tres algoritmos mencionados, confirmando así el funcionamiento adecuado del código.

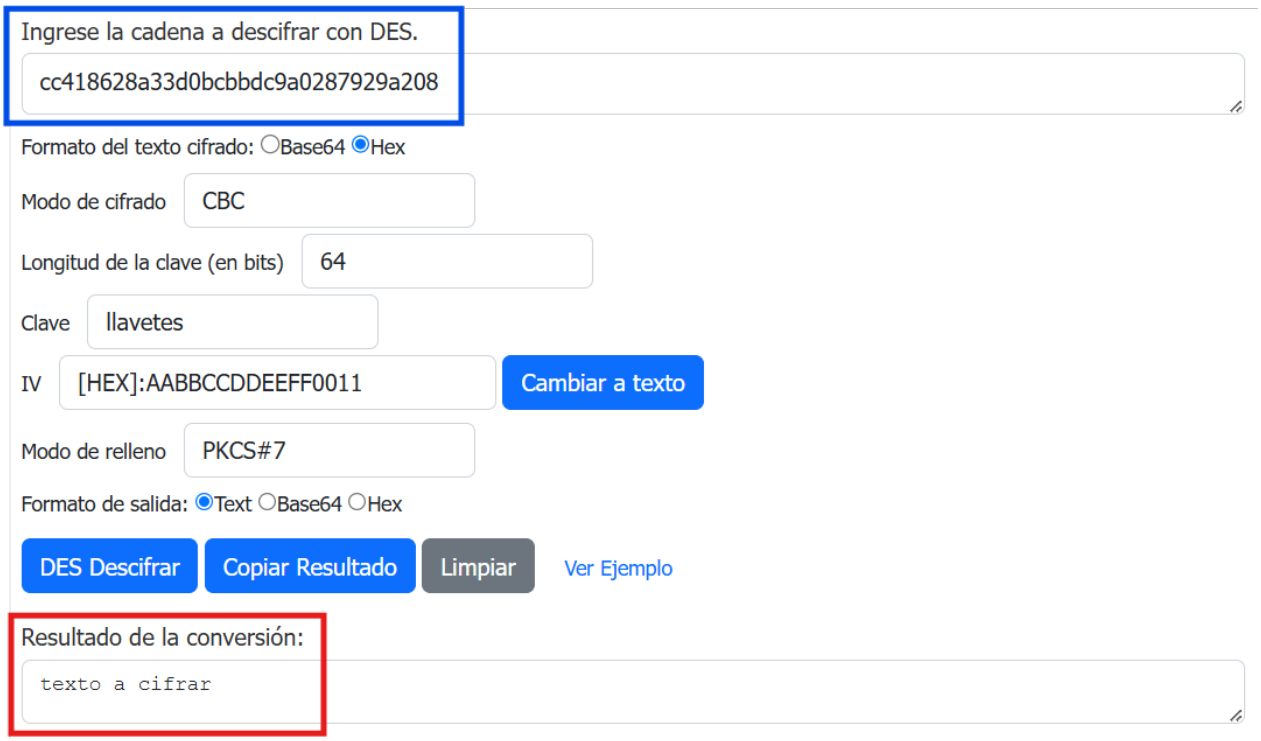

Figura 6: Pagina de decifrado del algoritmo DES

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

En un escenario donde la protección de la información sensible es crucial, como ocurre en las comunicaciones entre una empresa y sus proveedores para el intercambio de datos confidenciales, la implementación de cifrado simétrico resulta ser una opción adecuada. En este contexto, el algoritmo AES (Advanced Encryption Standard es altamente recomendable debido a su robustez y eficiencia. AES se distingue por su capacidad de ofrecer alta seguridad con claves de 128, 192 o 256 bits, lo que lo convierte en un estándar de facto en diversas aplicaciones de seguridad. Su desempeño optimizado en términos de velocidad y eficiencia lo hace ideal para entornos que requieren el procesamiento de grandes volúmenes de datos, garantizando la confidencialidad de la información durante su transmisión en protocolos de seguridad como SSL/TLS.

Por otro lado, en respuesta a la solicitud de reemplazar el cifrado simétrico por funciones de hash, es fundamental aclarar la distinción técnica entre ambos mecanismos. El cifrado simétrico se emplea para garantizar la confidencialidad de la información mediante un proceso reversible, permitiendo que los datos sean tanto cifrados como descifrados con la misma clave, lo cual es esencial en situaciones que requieren proteger datos sensibles. En cambio, las funciones de hash operan como algoritmos unidireccionales que generan un valor único (digest) para cualquier conjunto de datos, pero no permiten la recuperación del contenido original. Si bien los hash son útiles para la verificación de la integridad de los datos y la auten-

ticación de mensajes, no ofrecen protección contra el acceso no autorizado a la información. Por lo tanto, aunque las funciones de hash tienen su lugar en la seguridad de la información, el cifrado simétrico sigue siendo la opción más adecuada cuando la confidencialidad de los datos es una prioridad.

Conclusiones y comentarios

Luego de investigar en profundidad el funcionamiento de los algoritmos de cifrado simétrico DES, AES-256 y 3DES, se implementaron de manera funcional en un código en Python. Posteriormente, se empleó una página web para descifrar los mensajes generados y confirmar así la correcta encriptación de los mismos. Finalmente, se analizó la aplicación práctica del cifrado simétrico en contextos reales, como la protección de datos confidenciales en transmisiones y almacenamientos seguros. A través de este proceso, se logró cumplir con éxito el objetivo del laboratorio, adquiriendo una comprensión sólida de estos tres algoritmos fundamentales en la criptografía simétrica y sus aplicaciones. Esta experiencia ha permitido obtener un mayor conocimiento sobre cómo los algoritmos de cifrado simétrico aseguran la integridad y confidencialidad de la información en la vida cotidiana, reafirmando la relevancia de estos métodos en la seguridad de sistemas de información actuales.