

Informe Laboratorio 4

Sección 4

José Berríos Piña
e-mail: jose.berrios1@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	3
2.3. Valida y ajusta la clave según el algoritmo	4
2.4. Implementa el cifrado y descifrado en modo CBC	7
2.5. Compara los resultados con un servicio de cifrado online	16
2.6. Describe la aplicabilidad del cifrado simétrico en la vida real	17

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

1. DES

- **Tamaño de llave e IV:** La llave de DES es de 64 bits (8 bytes), pero solo se utilizan 56 bits (7 bytes), ya que 8 bits se usan para verificar la integridad (paridad). El vector de inicialización (IV) también es de 64 bits (8 bytes) y se usa en ciertos modos, como CBC y OFB, para asegurar que mensajes idénticos no se cifren igual, incluso con la misma llave, lo que aumenta la seguridad [1].
- **Características:** En comparación con otros algoritmos, DES es menos seguro debido a su llave más corta.

2. AES-256

- **Tamaño de llave e IV:** AES-256 usa una llave de 256 bits (32 bytes) y un IV de 128 bits (16 bytes) en el modo CBC. Su IV más largo mejora la seguridad al proporcionar mayor aleatoriedad [2].
- **Características:** AES-256 es más seguro y funciona más rápido en hardware adecuado, en comparación con DES y 3DES.

3. 3DES

- **Tamaño de llave e IV:** 3DES tiene una llave de 24 bytes (192 bits), aunque solo 21 bytes (168 bits) son efectivos debido a los bits de paridad. Además, debido a vulnerabilidades en DES, la seguridad efectiva de 3DES es de 14 bytes (112 bits). Su IV es de 64 bits (8 bytes) y ayuda a mejorar la seguridad en los modos que lo requieren, aunque sigue siendo menos eficiente que AES [3].
- **Características:** 3DES es una extensión de DES, pero es más lento y menos seguro que AES-256.

2.2. Solicita datos de entrada desde la terminal

Se creará un programa que contendrá los algoritmos de cifrado DES, AES-256 y 3DES. Cada cifrado estará contenido en sus respectivas funciones, y se utilizará la biblioteca *Cryptodome*, instalada mediante el comando *pip install pycryptodome*. Además, cada programa

recibirá la llave, el vector de inicialización y el mensaje a cifrar desde la terminal. Para facilitar el análisis, se utilizarán los mismos datos de entrada para todos los algoritmos, y el programa se encargará de rellenar o truncar los atributos de acuerdo con las necesidades de cada caso, tal como se solicita en el desarrollo del laboratorio.

Datos a ingresar para cada uno de los casos:

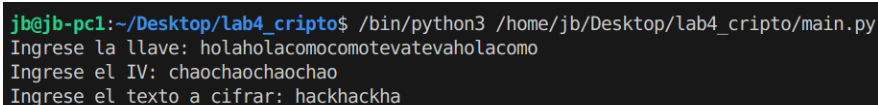
- **Llave:** holaholacomocomotevatevaholacomo
- **Vector de inicialización (IV):** chaochaochaochao
- **Mensaje:** hackhackha

```

1 # Entrada de datos
2 # llave_entrada = holaholacomocomotevatevaholacomo
3 llave_entrada = input("Ingrese la llave: ").encode()
4 # iv_entrada = chaochaochaochao
5 iv_entrada = input("Ingrese el IV: ").encode()
6 # texto_entrada = hackhackha
7 texto_entrada = input("Ingrese el texto a cifrar: ")
8
9 print("\n-----\n")

```

Bloque de código 1: Extracto de código encargado de solicitar los datos mediante la terminal



```

jbejb-pc1:~/Desktop/lab4_cripto$ /bin/python3 /home/jb/Desktop/lab4_cripto/main.py
Ingrese la llave: holaholacomocomotevatevaholacomo
Ingrese el IV: chaochaochaochao
Ingrese el texto a cifrar: hackhackha

```

Figura 1: Vista al funcionamiento de la terminal con el Bloque de código 1.

2.3. Valida y ajusta la clave según el algoritmo

1. DES

- Esta función recibe la llave proporcionada y ajusta su tamaño a 8 bytes. Si es menor, se completa con bytes aleatorios; si es mayor, se trunca. Lo mismo para IV, de 8 bytes.

```

1 from Crypto.Cipher import DES, DES3, AES
2 from Crypto.Random import get_random_bytes
3 import binascii
4
5 def preparar(valor, longitud_requerida, tipo):

```

```

6     if len(valor) < longitud_requerida:
7         valor += get_random_bytes(longitud_requerida - len(valor)
8             ))
9         print(f"{tipo}_ajustado_a_{longitud_requerida}_bytes_
10             anadiendo_bytes_adicionales.")
11     elif len(valor) > longitud_requerida:
12         valor = valor[:longitud_requerida]
13         print(f"{tipo}_excedia_{longitud_requerida}_bytes_y_fue_
14             truncado.")
15     # Imprimimos en Hexadecimal, facilitando la comprension y
16     comparativa.
17     print(f"{tipo}_final_utilizado_{hex}:", binascii.hexlify(
18         valor).decode())
19     return valor
20
21 # Ajustar llaves y IV segun el algoritmo
22 llave_DES = preparar(llave_entrada, 8, "Llave_DES")
23 iv_DES = preparar(iv_entrada, 8, "IV_DES")

```

Bloque de código 2: Extracto de código encargado de validar y ajustar los datos de DES.

```

Llave DES excedia 8 bytes y fue truncado.
Llave DES final utilizado (hex): 686f6c61686f6c61
IV DES excedia 8 bytes y fue truncado.
IV DES final utilizado (hex): 6368616f6368616f

```

Figura 2: Validación y ajuste de los datos del algoritmo DES.

2. AES-256

- Esta función recibe la llave proporcionada y ajusta su tamaño a 32 bytes. Si es menor, se completa con bytes aleatorios; si es mayor, se trunca. Lo mismo para IV, de 16 bytes.

```

1 from Crypto.Cipher import DES, DES3, AES
2 from Crypto.Random import get_random_bytes
3 import binascii
4
5 def preparar(valor, longitud_requerida, tipo):
6     if len(valor) < longitud_requerida:
7         valor += get_random_bytes(longitud_requerida - len(valor)
8             ))

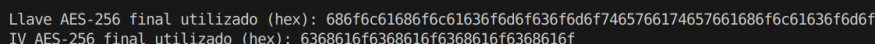
```

```

8         print(f"{tipo}_ajustado_a_{longitud_requerida}_bytes_
          anadiendo_bytes_adicionales.")
9     elif len(valor) > longitud_requerida:
10         valor = valor[:longitud_requerida]
11         print(f"{tipo}_excedia_{longitud_requerida}_bytes_y_fue_
          truncado.")
12     # Imprimimos en Hexadecimal, facilitando la comprension y
        comparativa.
13     print(f"{tipo}_final_utilizado_(hex):", binascii.hexlify(
        valor).decode())
14     return valor
15
16 llave_AES = preparar(llave_entrada, 32, "Llave_AES-256")
17 iv_AES = preparar(iv_entrada, 16, "IV_AES-256")

```

Bloque de código 3: Extracto de código encargado de validar y ajustar los datos de AES-256.



```

Llave AES-256 final utilizado (hex): 686f6c61686f6c61636f6d6f636f6d6f7465766174657661686f6c61636f6d6f
IV AES-256 final utilizado (hex): 6368616f6368616f6368616f6368616f

```

Figura 3: Validación y ajuste de los datos del algoritmo AES-256.

3. 3DES

- Esta función recibe la llave proporcionada y ajusta su tamaño a 24 bytes. Si es menor, se completa con bytes aleatorios; si es mayor, se trunca. Lo mismo para IV, de 8 bytes.

```

1 from Crypto.Cipher import DES, DES3, AES
2 from Crypto.Random import get_random_bytes
3 import binascii
4
5 def preparar(valor, longitud_requerida, tipo):
6     if len(valor) < longitud_requerida:
7         valor += get_random_bytes(longitud_requerida - len(valor))
8         print(f"{tipo}_ajustado_a_{longitud_requerida}_bytes_
          anadiendo_bytes_adicionales.")
9     elif len(valor) > longitud_requerida:
10         valor = valor[:longitud_requerida]
11         print(f"{tipo}_excedia_{longitud_requerida}_bytes_y_fue_
          truncado.")
12     # Imprimimos en Hexadecimal, facilitando la comprension y
        comparativa.

```

```

13     print(f"{tipo}_final_utilizado_(hex):", binascii.hexlify(
        valor).decode())
14     return valor
15
16 llave_3DES = preparar(llave_entrada, 24, "Llave_3DES")
17 iv_3DES = preparar(iv_entrada, 8, "IV_3DES")

```

Bloque de código 4: Extracto de código encargado de validar y ajustar los datos de 3DES.

```

Llave 3DES excedia 24 bytes y fue truncado.
Llave 3DES final utilizado (hex): 686f6c61686f6c61636f6d6f636f6d6f7465766174657661
IV 3DES excedia 8 bytes y fue truncado.
IV 3DES final utilizado (hex): 6368616f6368616f

```

Figura 4: Validación y ajuste de los datos del algoritmo 3DES.

2.4. Implementa el cifrado y descifrado en modo CBC

Una vez ingresados la llave, el IV y el texto a cifrar, el programa ajusta la llave y IV según sea necesario, cifra el mensaje y, posteriormente, lo descifra para mostrar el texto original.

1. DES

```

1 from Crypto.Cipher import DES, DES3, AES
2 from Crypto.Random import get_random_bytes
3 import binascii
4
5 # Cifra el texto utilizando DES en modo CBC
6 def cifrar_DES(llave, iv, texto):
7     # Creacion objeto cifrador para cifrar DES en modo CBC
8     cifrador = DES.new(llave, DES.MODE_CBC, iv)
9     # Padding (en caso de ser necesario) para cumplir la
10     longitud_multiplo_de_8 = 8 - (len(texto) % 8)
11     texto_rellenado = texto + (chr(longitud_relleno) *
12     longitud_relleno)
13     # Ciframos texto
14     texto_cifrado = cifrador.encrypt(texto_rellenado.encode())
15     # Imprime el texto cifrado en formato hexadecimal para
16     facilitar la comprension y comparativa
17     print("Texto_cifrado_con_DES_(hex):", binascii.hexlify(
18     texto_cifrado).decode())
19     # Retorna el texto cifrado en formato de bytes
20     return texto_cifrado

```

```
19 # Descifra el texto cifrado utilizando DES en modo CBC
20 def descifrar_DES(llave, iv, texto_cifrado):
21     # Creacion objeto cifrador para descifrar DES en modo CBC
22     cifrador = DES.new(llave, DES.MODE_CBC, iv)
23     # Se descifra el texto cifrado
24     texto_descifrado = cifrador.decrypt(texto_cifrado)
25     # Obtencion tamaño del relleno en base al ultimo byte del
        texto_descifrado
26     longitud_relleno = texto_descifrado[-1]
27     # Eliminacion del relleno
28     texto_descifrado = texto_descifrado[:-longitud_relleno].
        decode()
29     # Imprime el texto descifrado para verificar que coincida
        con el texto original antes de cifrar
30     print("Texto descifrado con DES:", texto_descifrado)
31     # Retorna el texto descifrado como cadena de texto
32     return texto_descifrado
33
34 # Cifrado y descifrado con cada algoritmo
35 print("\n--- Cifrado y Descifrado con DES ---")
36 texto_cifrado_DES = cifrar_DES(llave_DES, iv_DES, texto_entrada)
37 descifrar_DES(llave_DES, iv_DES, texto_cifrado_DES)
38
39 print("\n-----\n\n")
```

Bloque de código 5: Extracto de código encargado de cifrar y descifrar con el algoritmo DES.

```
--- Cifrado y Descifrado con DES ---
Texto cifrado con DES (hex): 5e50eb2190375c54c13860eb040872bc
Texto descifrado con DES: hackhackha
```

Figura 5: Cifrado y descifrado del algoritmo DES.

2. AES-256

```

1 from Crypto.Cipher import DES, DES3, AES
2 from Crypto.Random import get_random_bytes
3 import binascii
4
5 # Cifra el texto utilizando AES-256 en modo CBC
6 def cifrar_AES(llave, iv, texto):
7     # Creacion objeto cifrador para cifrar AES-256 en modo CBC
8     cifrador = AES.new(llave, AES.MODE_CBC, iv)
9     # Padding (en caso de ser necesario) para cumplir la
10     longitud_relleno = 16 - (len(texto) % 16)
11     texto_rellenado = texto + (chr(longitud_relleno) *
12     longitud_relleno)
13     # Ciframos texto
14     texto_cifrado = cifrador.encrypt(texto_rellenado.encode())
15     # Imprime el texto cifrado en formato hexadecimal para
16     facilitar la comprension y comparativa
17     print("Texto_cifrado_con_AES-256(hex):", binascii.hexlify(
18     texto_cifrado).decode())
19     # Retorna el texto cifrado en formato de bytes
20     return texto_cifrado
21
22 # Descifra el texto cifrado utilizando AES-256 en modo CBC
23 def descifrar_AES(llave, iv, texto_cifrado):
24     # Creacion objeto cifrador para descifrar AES-256 en modo
25     CBC
26     cifrador = AES.new(llave, AES.MODE_CBC, iv)
27     # Se descifra el texto cifrado
28     texto_descifrado = cifrador.decrypt(texto_cifrado)
29     # Obtencion tamano del relleno en base al ultimo byte del
30     texto descifrado
31     longitud_relleno = texto_descifrado[-1]
32     # Eliminacion del relleno
33     texto_descifrado = texto_descifrado[:-longitud_relleno].
34     decode()
35     # Imprime el texto descifrado para verificar que coincida
36     con el texto original antes de cifrar
37     print("Texto_descifrado_con_AES-256:", texto_descifrado)
38     # Retorna el texto descifrado como cadena de texto
39     return texto_descifrado
40
41 print("\n---_Cifrado_y_Descifrado_con_AES-256_---")
42 texto_cifrado_AES = cifrar_AES(llave_AES, iv_AES, texto_entrada)
43 descifrar_AES(llave_AES, iv_AES, texto_cifrado_AES)

```

```

37
38 print("\n-----\n\n")

```

Bloque de código 6: Extracto de código encargado de cifrar y descifrar con el algoritmo AES-256.

```

--- Cifrado y Descifrado con AES-256 ---
Texto cifrado con AES-256 (hex): 8895e224fda98a27a256dc3ebf8be17e
Texto descifrado con AES-256: hackhackha

```

Figura 6: Cifrado y descifrado del algoritmo AES-256.

3. 3DES

```

1 from Crypto.Cipher import DES, DES3, AES
2 from Crypto.Random import get_random_bytes
3 import binascii
4
5 # Cifra el texto utilizando 3DES en modo CBC
6 def cifrar_3DES(llave, iv, texto):
7     # Creacion objeto cifrador para cifrar 3DES en modo CBC
8     cifrador = DES3.new(llave, DES3.MODE_CBC, iv)
9     # Padding (en caso de ser necesario) para cumplir la
10     longitud multiplo de 8 (bloque de 3DES)
11     longitud_relleno = 8 - (len(texto) % 8)
12     texto_rellenado = texto + (chr(longitud_relleno) *
13     longitud_relleno)
14     # Ciframos texto
15     texto_cifrado = cifrador.encrypt(texto_rellenado.encode())
16     # Imprime el texto cifrado en formato hexadecimal para
17     facilitar la comprension y comparativa
18     print("Texto_cifrado_con_3DES(hex):", binascii.hexlify(
19     texto_cifrado).decode())
20     # Retorna el texto cifrado en formato de bytes
21     return texto_cifrado
22
23 # Descifra el texto cifrado utilizando 3DES en modo CBC
24 def descifrar_3DES(llave, iv, texto_cifrado):
25     # Creacion objeto cifrador para descifrar 3DES en modo CBC
26     cifrador = DES3.new(llave, DES3.MODE_CBC, iv)
27     # Se descifra el texto cifrado
28     texto_descifrado = cifrador.decrypt(texto_cifrado)
29     # Obtencion tamaño del relleno en base al ultimo byte del
30     texto descifrado

```

```

26     longitud_relleno = texto_descifrado[-1]
27     # Eliminacion del relleno
28     texto_descifrado = texto_descifrado[:-longitud_relleno].
        decode()
29     # Imprime el texto descifrado para verificar que coincida
        con el texto original antes de cifrar
30     print("Texto_descifrado_con_3DES:", texto_descifrado)
31     # Retorna el texto descifrado como cadena de texto
32     return texto_descifrado
33
34 print("\n---_Cifrado_y_Descifrado_con_3DES_---")
35 texto_cifrado_3DES = cifrar_3DES(llave_3DES, iv_3DES,
        texto_entrada)
36 descifrar_3DES(llave_3DES, iv_3DES, texto_cifrado_3DES)

```

Bloque de código 7: Extracto de código encargado de cifrar y descifrar con el algoritmo 3DES.

```

--- Cifrado y Descifrado con 3DES ---
Texto cifrado con 3DES (hex): 52e8fc53177164df7eec8c0bcc591377
Texto descifrado con 3DES: hackhackha

```

Figura 7: Cifrado y descifrado del algoritmo 3DES.

Notar que la librería PyCryptodome y sus ejemplos facilitan bastante la escritura de códigos [4].

A continuación, el bloque que contiene el código completo solicitado:

```

1 from Crypto.Cipher import DES, DES3, AES
2 from Crypto.Random import get_random_bytes
3 import binascii
4
5 def preparar(valor, longitud_requerida, tipo):
6     if len(valor) < longitud_requerida:
7         valor += get_random_bytes(longitud_requerida - len(valor))
8         print(f"{tipo}_ajustado_a_{longitud_requerida}_bytes_
            anadiendo_bytes_adicionales.")
9     elif len(valor) > longitud_requerida:
10        valor = valor[:longitud_requerida]
11        print(f"{tipo}_excedia_{longitud_requerida}_bytes_y_fue_
            truncado.")
12    # Imprimimos en Hexadecimal, facilitando la comprension y
        comparativa.

```

```
13     print(f"{tipo}_final_utilizado_(hex):", binascii.hexlify(valor).  
14           decode())  
15     return valor  
16  
17 # Cifra el texto utilizando DES en modo CBC  
18 def cifrar_DES(llave, iv, texto):  
19     # Creacion objeto cifrador para cifrar DES en modo CBC  
20     cifrador = DES.new(llave, DES.MODE_CBC, iv)  
21     # Padding (en caso de ser necesario) para cumplir la longitud  
22     # multiplo de 8 (bloque de DES)  
23     longitud_relleno = 8 - (len(texto) % 8)  
24     texto_rellenado = texto + (chr(longitud_relleno) *  
25         longitud_relleno)  
26     # Ciframos texto  
27     texto_cifrado = cifrador.encrypt(texto_rellenado.encode())  
28     # Imprime el texto cifrado en formato hexadecimal para facilitar  
29     # la comprension y comparativa  
30     print("Texto_cifrado_con_DES_(hex):", binascii.hexlify(  
31         texto_cifrado).decode())  
32     # Retorna el texto cifrado en formato de bytes  
33     return texto_cifrado  
34  
35 # Descifra el texto cifrado utilizando DES en modo CBC  
36 def descifrar_DES(llave, iv, texto_cifrado):  
37     # Creacion objeto cifrador para descifrar DES en modo CBC  
38     cifrador = DES.new(llave, DES.MODE_CBC, iv)  
39     # Se descifra el texto cifrado  
40     texto_descifrado = cifrador.decrypt(texto_cifrado)  
41     # Obtencion tamaño del relleno en base al ultimo byte del texto  
42     # descifrado  
43     longitud_relleno = texto_descifrado[-1]  
44     # Eliminacion del relleno  
45     texto_descifrado = texto_descifrado[:-longitud_relleno].decode()  
46     # Imprime el texto descifrado para verificar que coincida con el  
47     # texto original antes de cifrar  
48     print("Texto_descifrado_con_DES:", texto_descifrado)  
49     # Retorna el texto descifrado como cadena de texto  
50     return texto_descifrado  
51  
52 # Cifra el texto utilizando AES-256 en modo CBC  
53 def cifrar_AES(llave, iv, texto):  
54     # Creacion objeto cifrador para cifrar AES-256 en modo CBC  
55     cifrador = AES.new(llave, AES.MODE_CBC, iv)  
56     # Padding (en caso de ser necesario) para cumplir la longitud
```

```

    multiplo de 16 (bloque de AES)
52 longitud_relleno = 16 - (len(texto) % 16)
53 texto_rellenado = texto + (chr(longitud_relleno) *
    longitud_relleno)
54 # Ciframos texto
55 texto_cifrado = cifrador.encrypt(texto_rellenado.encode())
56 # Imprime el texto cifrado en formato hexadecimal para facilitar
    la comprension y comparativa
57 print("Texto_cifrado_con_AES-256(hex):", binascii.hexlify(
    texto_cifrado).decode())
58 # Retorna el texto cifrado en formato de bytes
59 return texto_cifrado
60
61 # Descifra el texto cifrado utilizando AES-256 en modo CBC
62 def descifrar_AES(llave, iv, texto_cifrado):
63     # Creacion objeto cifrador para descifrar AES-256 en modo CBC
64     cifrador = AES.new(llave, AES.MODE_CBC, iv)
65     # Se descifra el texto cifrado
66     texto_descifrado = cifrador.decrypt(texto_cifrado)
67     # Obtencion tamano del relleno en base al ultimo byte del texto
        descifrado
68     longitud_relleno = texto_descifrado[-1]
69     # Eliminacion del relleno
70     texto_descifrado = texto_descifrado[:-longitud_relleno].decode()
71     # Imprime el texto descifrado para verificar que coincida con el
        texto original antes de cifrar
72     print("Texto_descifrado_con_AES-256:", texto_descifrado)
73     # Retorna el texto descifrado como cadena de texto
74     return texto_descifrado
75
76 # Cifra el texto utilizando 3DES en modo CBC
77 def cifrar_3DES(llave, iv, texto):
78     # Creacion objeto cifrador para cifrar 3DES en modo CBC
79     cifrador = DES3.new(llave, DES3.MODE_CBC, iv)
80     # Padding (en caso de ser necesario) para cumplir la longitud
        multiplo de 8 (bloque de 3DES)
81     longitud_relleno = 8 - (len(texto) % 8)
82     texto_rellenado = texto + (chr(longitud_relleno) *
        longitud_relleno)
83     # Ciframos texto
84     texto_cifrado = cifrador.encrypt(texto_rellenado.encode())
85     # Imprime el texto cifrado en formato hexadecimal para facilitar
        la comprension y comparativa
86     print("Texto_cifrado_con_3DES(hex):", binascii.hexlify(
        texto_cifrado).decode())
87     # Retorna el texto cifrado en formato de bytes

```

```

88     return texto_cifrado
89
90 # Descifra el texto cifrado utilizando 3DES en modo CBC
91 def descifrar_3DES(llave, iv, texto_cifrado):
92     # Creacion objeto cifrador para descifrar 3DES en modo CBC
93     cifrador = DES3.new(llave, DES3.MODE_CBC, iv)
94     # Se descifra el texto cifrado
95     texto_descifrado = cifrador.decrypt(texto_cifrado)
96     # Obtencion tamano del relleno en base al ultimo byte del texto
97     longitud_relleno = texto_descifrado[-1]
98     # Eliminacion del relleno
99     texto_descifrado = texto_descifrado[:-longitud_relleno].decode()
100    # Imprime el texto descifrado para verificar que coincida con el
101    texto original antes de cifrar
102    print("Texto_descifrado_con_3DES:", texto_descifrado)
103    # Retorna el texto descifrado como cadena de texto
104    return texto_descifrado
105
106
107 # Entrada de datos
108 # llave_entrada = holaholacomocomotevatevaholacomo
109 llave_entrada = input("Ingrese_la_llave: ").encode()
110 # iv_entrada = chaochaochaochao
111 iv_entrada = input("Ingrese_el_IV: ").encode()
112 # texto_entrada = hackhackha
113 texto_entrada = input("Ingrese_el_texto_a_cifrar: ")
114
115 print("\n-----\n")
116
117
118 # Ajustar llaves y IV segun el algoritmo
119 llave_DES = preparar(llave_entrada, 8, "Llave_DES")
120 iv_DES = preparar(iv_entrada, 8, "IV_DES")
121 # Cifrado y descifrado con cada algoritmo
122 print("\n---Cifrado_y_Descifrado_con_DES---")
123 texto_cifrado_DES = cifrar_DES(llave_DES, iv_DES, texto_entrada)
124 descifrar_DES(llave_DES, iv_DES, texto_cifrado_DES)
125
126 print("\n-----\n\n")
127
128
129 llave_AES = preparar(llave_entrada, 32, "Llave_AES-256")
130 iv_AES = preparar(iv_entrada, 16, "IV_AES-256")
131 print("\n---Cifrado_y_Descifrado_con_AES-256---")

```

```

132 texto_cifrado_AES = cifrar_AES(llave_AES, iv_AES, texto_entrada)
133 descifrar_AES(llave_AES, iv_AES, texto_cifrado_AES)
134
135 print("\n-----\n\n")
136
137 llave_3DES = preparar(llave_entrada, 24, "Llave_3DES")
138 iv_3DES = preparar(iv_entrada, 8, "IV_3DES")
139 print("\n---_Cifrado_y_Descifrado_con_3DES_---")
140 texto_cifrado_3DES = cifrar_3DES(llave_3DES, iv_3DES, texto_entrada)
141 descifrar_3DES(llave_3DES, iv_3DES, texto_cifrado_3DES)

```

Bloque de código 8: Código completo algoritmos de cifrado.

Parámetro	Valor
Llave Ingresada	holaholacomocomotevatevaholacomo
IV Ingresado	chaochaochaochao
Texto a Cifrar	hackhackha

Tabla 1: Datos de Entrada para Bloque de código 8.

Parámetro	Valor
Llave DES Final Utilizada	686f6c61686f6c61 (truncada a 8 bytes)
IV DES Final Utilizado	6368616f6368616f (truncada a 8 bytes)
Texto Cifrado con DES (hex)	5e50eb2190375c54c13860eb040872bc
Texto Descifrado con DES	hackhackha

Tabla 2: Proceso de Cifrado y Descifrado con DES para Bloque de código 8.

Parámetro	Valor
Llave AES-256 Final Utilizada	686f6c61686f6c61636f6d6f636f6d6f7465766174657661686f6c61636f6d6f (32 bytes)
IV AES-256 Final Utilizado	6368616f6368616f6368616f6368616f (16 bytes)
Texto Cifrado con AES-256 (hex)	8895e224fda98a27a256dc3ebf8be17e
Texto Descifrado con AES-256	hackhackha

Tabla 3: Proceso de Cifrado y Descifrado con AES-256 para Bloque de código 8.

Parámetro	Valor
Llave 3DES Final Utilizada	686f6c61686f6c61636f6d6f636f6d6f7465766174657661 (truncada a 24 bytes)
IV 3DES Final Utilizado	6368616f6368616f (truncada a 8 bytes)
Texto Cifrado con 3DES (hex)	52e8fc53177164df7eec8c0bcc591377
Texto Descifrado con 3DES	hackhackha

Tabla 4: Proceso de Cifrado y Descifrado con 3DES para Bloque de código 8.

2.5. Compara los resultados con un servicio de cifrado online

Para esta sección se utilizara el servicio de cifrado AES-256 [6].

Los datos a ingresar son:

- **Llave:** holaholacomocomotevatevaholacom
- **Vector de inicializacion (IV):** chaochaochaochao
- **Mensaje:** hackhackha

Figura 8: Resultado del cifrado con el servicio online AES Encryption, configurado para AES-256 [6].

```

--- Cifrado y Descifrado con AES-256 ---
Texto cifrado con AES-256 (hex): 8895e224fda98a27a256dc3ebf8be17e
Texto descifrado con AES-256: hackhackha

```

Figura 9: Resultado del cifrado con los datos anteriores usando el algoritmo AES-256.

- **Resultado de codigo python AES-256:** 8895e224fda98a27a256dc3ebf8be17e.
- **Resultado de la pagina con AES-256:** 8895e224fda98a27a256dc3ebf8be17e.

Como se puede ver, en ambos casos se obtiene el mismo resultado, ya que se utiliza exactamente la misma llave, IV y mensaje para cifrar.

En otras páginas he realizado pruebas, y las variaciones que he encontrado en los resultados se deben principalmente al tipo de padding seleccionado para el algoritmo AES-256 que cada página utiliza.

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

Suponiendo un caso en el que se manejan datos sensibles, como en un banco o un centro médico, donde la información personal o financiera es delicada se recomienda el siguiente algoritmo de cifrado simétrico:

AES-256, con una llave de 256 bits, ofrece una cantidad inmensa de combinaciones posibles, es decir, 2^{256} combinaciones, lo que dificulta los ataques de fuerza bruta a un nivel casi imposible de resolver en la actualidad. Además, AES-256 opera con 14 rondas de cifrado que utilizan operaciones matemáticas complejas sobre los bits, de modo que un ligero cambio en el texto original genera resultados completamente diferentes [7]. Hasta el momento, no existen vulnerabilidades prácticas comprobadas de este algoritmo que lo hagan susceptible a ataques, destacando su resistencia contra técnicas avanzadas. AES-256 también es eficiente, ya que aprovecha el hardware y software disponibles en la actualidad, permitiendo manejar grandes cantidades de datos sin retrasos notables. Por último, AES-256 ha sido aprobado por el NIST y está incluido en estándares de cifrado internacionales, como el estándar de seguridad de datos de la industria de tarjetas de pago (PCI-DSS) y el Reglamento General de Protección de Datos (GDPR) de la Unión Europea [7][8][9].

Ahora, suponiendo que el cliente desea implementar *hashes* en lugar del cifrado y descifrado recomendados, tendríamos un buen nivel de seguridad y cierta facilidad de implementación. Aun así, el uso de hashes tiene el problema de que, si se filtra la información sensible de la función de hashing (como la semilla, longitud, etc.), podría replicarse fácilmente. En caso de filtrarse, sería necesario eliminar la información para proteger los datos, ya que los datos hashados no se pueden descifrar, a diferencia de los algoritmos de cifrado que se han estudiado en este informe. Además, implementar hashes limita considerablemente el uso de los datos, ya que complica la utilización de tipos de datos que no sean solo nombres de usuario o contraseñas.

Conclusiones y comentarios

Este laboratorio nos permitió probar los algoritmos DES, AES-256 y 3DES utilizando Python y la biblioteca PyCryptodome. Cada uno de estos algoritmos presenta particularidades en cuanto a la longitud de la llave y el IV, y su elección depende de factores como la velocidad y el nivel de seguridad deseado. Sin embargo, es importante notar que tanto DES como 3DES están actualmente vulnerados.

El proceso de ajustar las llaves al tamaño requerido para cada algoritmo y de garantizar la configuración correcta del vector de inicialización (IV) fue tedioso y requirió un esfuerzo adicional. La investigación previa en diversas fuentes y documentación, principalmente de la propia biblioteca, resultó crucial para entender el funcionamiento de estos algoritmos.

Obtener cifrados y descifrados consistentes fue fundamental durante la actividad, en la cual también se verificó el funcionamiento con distintos servicios en línea. Esto permitió una mayor comprensión de los parámetros que utilizan los algoritmos y cómo un ligero cambio puede afectar considerablemente el resultado. Este aspecto es relevante, ya que permite observar el conocido 'efecto avalancha', deseable y crucial en los algoritmos criptográficos.

Este laboratorio evidenció que la elección del algoritmo de cifrado está directamente ligada al nivel de seguridad requerido en cada caso. Además, es importante destacar que, con el tiempo, surgen nuevos algoritmos más eficientes y con un mayor nivel de seguridad en comparación con los antiguos.

Referencias

- [1] PyCryptodome Documentation. *Single DES*. Disponible en: <https://pycryptodome.readthedocs.io/en/latest/src/cipher/des.html> [Accedido el 5 de noviembre de 2024]. 2024.
- [2] PyCryptodome Documentation. *AES Cipher in PyCryptodome*. Disponible en: <https://pycryptodome.readthedocs.io/en/latest/src/cipher/aes.html> [Accedido el 5 de noviembre de 2024]. 2024.
- [3] PyCryptodome Documentation. *Triple DES (3DES) en PyCryptodome*. Disponible en: <https://pycryptodome.readthedocs.io/en/latest/src/cipher/des3.html> [Accedido el 5 de noviembre de 2024]. 2024.
- [4] PyCryptodome Developers. *PyCryptodome Documentation*. Accessed: 2024-11-12. 2024. URL: <https://pycryptodome.readthedocs.io/en/latest/src/examples.html>.
- [5] OpenAI. *ChatGPT*. Modelo de lenguaje grande (IA) desarrollado por OpenAI. 2024. URL: <https://openai.com/chatgpt>.
- [6] Devglan. *AES Encryption and Decryption Online Tool*. Accedido el 06 de noviembre de 2024. 2023. URL: <https://www.devglan.com/online-tools/aes-encryption-decryption>.
- [7] National Institute of Standards y Technology. “FIPS 197: Advanced Encryption Standard (AES)”. En: (2001). Accedido el 06 de noviembre de 2024. URL: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf>.
- [8] Payment Card Industry Security Standards Council. “PCI Data Security Standard (PCI DSS)”. En: (2024). Accedido el 06 de noviembre de 2024. URL: https://www.pcisecuritystandards.org/pci_security/.
- [9] European Union. “Reglamento (UE) 2016/679 del Parlamento Europeo y del Consejo”. En: (2016). Accedido el 06 de noviembre de 2024. URL: <https://eur-lex.europa.eu/eli/reg/2016/679/oj>.