Informe Laboratorio 4

Sección 1

Alumno Diego Banda
e-mail: diego.banda@mail.udp.cl

Noviembre de 2025

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

Según la página '*geek for geeks - Data Encryption Standard (DES)*', tal algoritmo encripta información por bloques de 64 bits, retornando la información encriptada también en 64 bits, utiliza una clave (o *key*) de 64 bits, sin embargo, a esta se le quita cada 8vo bit de cada byte, resultando en una cadena de 56 bits efectiva de utilización.

Por otro lado, *3DES* fue la 'evolución' de *DES*, ya que es un algoritmo más seguro al aplicar *DES* 3 veces a una cadena y a la vez fácil de aplicar teniendo ya la implementación de su predecesor, según el blog '*X sec - 3DES*' del autor Nestor Guerra, se sigue encriptando en bloques de 64 bits, sin embargo ahora que se estará encriptando 3 veces, se utilizarán 3 contraseñas también, en el blog se menciona uso de claves efectivas de 112 y 168, las cuales son el resultado de tener 2 y 3 claves efectivas de 56 bits respectivamente, y si tomamos la clave antes de quitar cada 8vo bit, sería entonces una clave de 128 y 192. Se tienen 3 claves, si se usan las 3 claves iguales, se tendrá el mismo algoritmo que *DES* normal, por otro lado, si se tiene la primera con la última clave iguales, este sería una variante llamada '*Two-key 3DES*' la cual tendría la clave efectiva de 112 o clave total de 128, y finalmente si se tienen las 3 claves distintas, es estarían ocupando 168 bits para la clave efectiva y 192 para largo total de clave.

Y el último algoritmo de cifrado a analizar es *AES-256* (*Advanced Encryption Standard*), el cual es el que presenta mayor nivel de seguridad contra ataques de fuerza bruta, encripta en bloques de 128 bits, y cuenta con 3 versiones para la clave, *AES-128*, *AES-192* y *AES-256*, sin embargo el estudiado es el último, contando con una clave de 256 bits (referencia: 'Everything You Need to Know About AES-256 Encryption - Kiteworks').

En cuanto a los vectores de inicialización (inicialization vector, IV) son secuencias de bits para añadir aleatoriedad al proceso de cifrado, de esta manera se asegura mayor seguridad para el servicio, cada *IV* debe ser único para garantizar que no se identifiquen patrones en los resultados de cifrados. El *IV* se aplica cuando estamos trabajando con modo *CBC*, el tamaño que tiene dependerá del cifrado utilizado, pero más específico, del tamaño del bloque a cifrar de estos, entonces, para *DES* y *3DES* que cifran en bloques de 64 bits su *IV* es de 64 bits (8 bytes) y para *AES-256* que cifra en bloques de 128 bits su *IV* tendrá un tamaño de 128 bits (16 bytes).

	Largo bloque encriptado	**Largo clave**	**Largo *IV***
DES	8 bytes	8 bytes	8 bytes
3DES	8 bytes	16 o 24 bytes	8 bytes
AES-256	16 bytes	32 bytes	16 bytes

Tabla 1: Tabla comparativa

Entre *DES* y *3DES* la principal diferencia es que el segundo es mucho más seguro, pero ineficiente y lento computacionalmente, por otro lado está también *AES-256* el cual es muy seguro y el estándar actual, ya que también es rápido de computar, combinando buenas cualidades, otra diferencia es en los bytes que encriptan, los primeros dos encriptando tan solo 8 bytes, mientras que *AES-256* 16 bytes, o el largo de sus claves, creciendo a medida que se añade mayor seguridad al encriptado.

2.2. Solicita datos de entrada desde la terminal

```
print("==== DES ====")
DES_key = input("Key para DES (8 bytes): ")
DES_IV = input("IV para DES (8 bytes): ")
IV_verification(DES_IV, 1)
print("==== 3DES ====")
DES3_key = input("Key para 3DES (24 bytes): ")
DES3_IV = input("IV para 3DES (8 bytes): ")
IV_verification(DES3_IV, 2)
print("==== AES ====")
AES256_key = input("Key para AES-256 (32 bytes): ")
AES256_IV = input("IV para AES-256 (16 bytes): ")
IV_verification(AES256_IV, 3)
print("==== Texto ====")
text = input("Texto a cifrar: ")
```

Listing 1: Ingreso de datos

Con esta sección de código se pide a través de los *inputs* el ingreso de los datos correspondientes a cada cifrado, de esta manera el usuario hace ingreso de todo lo necesario para aplicarlos.

```
> python3 code.py
==================
===== Ingreso =====
==================

==== DES ====
Key para DES (8 bytes): colocolo
IV para DES (8 bytes): colocolo
==== 3DES ====
Key para 3DES (24 bytes): colocolocampeonlibertadores
IV para 3DES (8 bytes): colocolo
==== AES ====
Key para AES-256 (32 bytes): colocolocampeondelibertadores1991
IV para AES-256 (16 bytes): colocolocampeon1
==== Texto ====
Texto a cifrar: ola
```

Figura 1: Ingreso Correcto

Cómo se puede ver en la imagen, los ingresos se piden con una serie de *prints* para saber qué se está ingresando y funciona correctamente, por otro lado también se está llamando a la función '*IV_verification*', la cual recibe el vector de iniciación y el algoritmo de cifrado correspondiente a través de un número, esta función verifica que exista la cantidad de bytes justos para realizar el cifrado, de caso de no cumplir, se termina el código a través de un *ValueError*.

```
1 def IV_verification(IV, opt):
2     IV = IV.encode("utf-8")
3     length = len(IV)
4     if (length != 8 and (opt == 1 or opt == 2)) or (length != 16 and opt
      == 3):
5         raise ValueError("Largo de IV ingresado incorrecto!")
```

Listing 2: Función: *IV_verification*

```
==== DES ====
Key para DES (8 bytes): colocolo
IV para DES (8 bytes): 6bytes
Traceback (most recent call last):
  File "/Users/diego/Desktop/Criptografia/Lab04/code.py", line 70, in <module>
    IV_verification(DES_IV, 1)
  File "/Users/diego/Desktop/Criptografia/Lab04/code.py", line 34, in IV_verification
    raise ValueError("Largo de IV ingresado incorrecto!")
ValueError: Largo de IV ingresado incorrecto!
```

Figura 2: Ingreso incorrecto de *IV*

2.3. Valida y ajusta la clave según el algoritmo

Para asegurarse que se está usando la clave en el tamaño correcto, para cada una se llama a la siguiente función:

```python
def to_bytes_config(txt, num_bytes):
    txt = txt.encode("utf-8")
    length = len(txt)
    if length > num_bytes:
        txt = txt[:num_bytes]
    elif length < num_bytes:
        txt = txt + get_random_bytes(num_bytes - length)
    return txt
```

Listing 3: Función: *to_bytes_config*

Esta función recibe como parámetro la *key* y el numero de bytes que debe tener, en caso de tener más, se trunca y en caso de faltar, se agregan bytes aleatorios. Esta función se lama con cada una de las *key* correspondientes a cada cifrado, para luego imprimir la contraseña que se aplicará, ya sea la original, o esté truncada o con bytes extras.

```python
#   DES
DES_key = to_bytes_config(DES_key, 8)
print(f"Clave a utilizar para DES (bytes): {DES_key}")
#   3DES
DES3_key = to_bytes_config(DES3_key, 24)
print(f"Clave a utilizar para 3DES (bytes): {DES3_key}")
#   AES256
AES256_key = to_bytes_config(AES256_key, 32)
print(f"Clave a utilizar para AES-256 (bytes): {AES256_key}")
```

Listing 4: Llamado a función para cada *key*

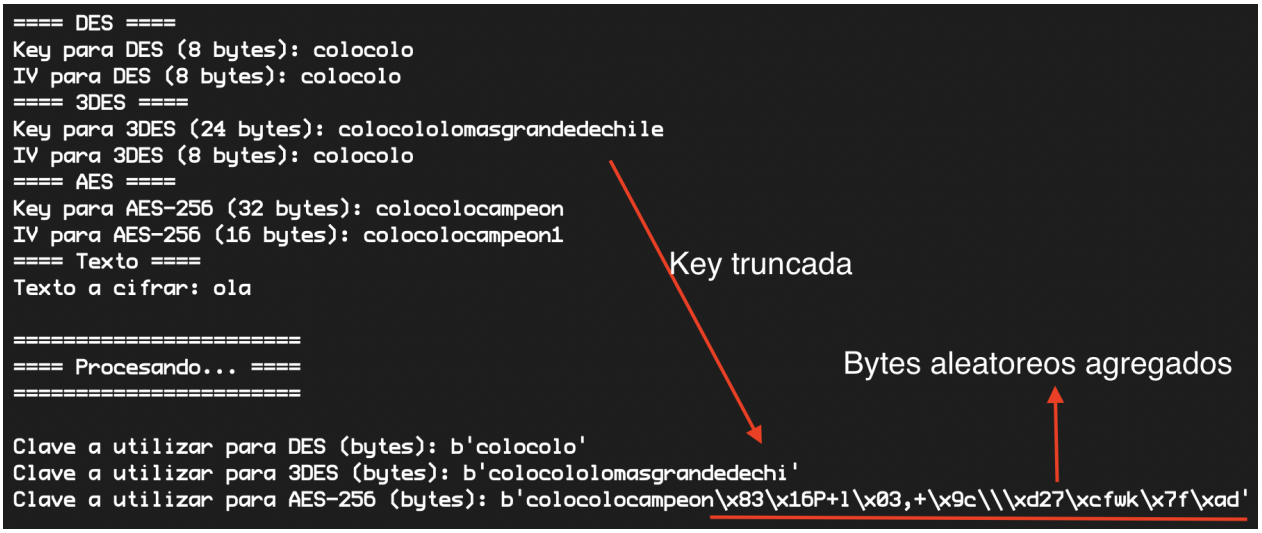

Figura 3: Ejemplo de truncar y *get_random_bytes* en *keys*

2.4. Implementa el cifrado y descifrado en modo CBC

```
#   DES
DES_encrypted = DES_encrypt(text, DES_key, DES_IV)
DES_encrypted_b64 = base64.b64encode(DES_encrypted).decode("utf-8")
print(f"Texto encriptado con DES (base64): {DES_encrypted_b64}")
print(f"Texto encriptado con DES (HEX): {DES_encrypted.hex()}\n")
#   3DES
DES3_encrypted = DES3_encrypt(text, DES3_key, DES3_IV)
DES3_encrypted_b64 = base64.b64encode(DES3_encrypted).decode("utf-8")
print(f"Texto encriptado con 3DES (base64): {DES3_encrypted_b64}")
print(f"Texto encriptado con 3DES (HEX): {DES3_encrypted.hex()}\n")
#   AES256
AES256_encrypted = AES256_encrypt(text, AES256_key, AES256_IV)
AES256_encrypted_b64 = base64.b64encode(AES256_encrypted).decode("utf-8")
print(f"Texto encriptado con AES-256 (base64): {AES256_encrypted_b64}")
print(f"Texto encriptado con AES-256 (HEX): {AES256_encrypted.hex()}")
```

Listing 5: Llamado a función y *print* para cada encriptación

Como se observa en el código, para cada cifrado se implementó una función propia (las cuales se puede observar más a detalle en el siguiente bloque de código), una vez se tiene el texto encriptado para cada algoritmo, se imprime el resultado, mostrándolo en pantalla con dos versiones, una en *base64* y otra en hexadecimal.

```
def DES_encrypt(txt, key, IV):
    txt = txt.encode("utf-8")
    enc = DES.new(key, DES.MODE_CBC, IV)
    txt = pad(txt, DES.block_size)
    return enc.encrypt(txt)

def DES3_encrypt(txt, key, IV):
    txt = txt.encode("utf-8")
    enc = DES3.new(key, DES3.MODE_CBC, IV)
    txt = pad(txt, DES3.block_size)
    return enc.encrypt(txt)

def AES256_encrypt(txt, key, IV):
    txt = txt.encode("utf-8")
    enc = AES.new(key, AES.MODE_CBC, IV)
    txt = pad(txt, AES.block_size)
    return enc.encrypt(txt)
```

Listing 6: Funciones de encriptado

Esta parte del código fue probablemente la más difícil, pero más que por el código, fue por la versión tan antigua de *pycrypto*, para instalarla y utilizarla correctamente había que seguir muchos pasos, pero una vez instalada, se realizan estas funciones, las cuales reciben el texto a encriptar y sus correspondientes *key* e *IV* ingresadas por el usuario anteriormente, para luego convertir a bytes el texto (para poder manejarlo), luego se crea el objeto que encriptará bajo tal técnica, en esta creación se le entrega la *key* a utilizar, se específica el modo de encriptado (en este caso *CBC*) y el *IV*, luego por la versión tan antigua de la librería, se tuvo que

implementar una función de *padding* necesaria para cumplir con los bytes a encriptar del texto, finalmente con el *padding* agregado se retorna el texto encriptado.

```
========================
==== Encriptando... ====
========================

Texto encriptado con DES (base64): yQHGICg70iw=
Texto encriptado con DES (HEX): c901c620283bd22c

Texto encriptado con 3DES (base64): dYj7lgtD604=
Texto encriptado con 3DES (HEX): 7588fb960b43eb4e

Texto encriptado con AES-256 (base64): qAApbqEzwq4kx01eErAkGA==
Texto encriptado con AES-256 (HEX): a800296ea133c2ae24c74d5e12b02418
```

Figura 4: Texto encriptado

La función de *padding* funciona bajo el estándar *PKCS#7*, el cual básicamente añade los bytes faltantes para llegar al tamaño de bloque que encripta el algoritmo en cuestión, los bytes añadidos tienen en sí la cantidad de bytes que son agregados de *padding*, de esta manera se pueden identificar y eliminar después, en caso de no necesitar agregar, igualmente se agregan con la cantidad correspondiente a un bloque completo, ya que el *unpadding* se ejecutará igualmente y el no agregarlo podría generar errores.

```python
def pad(txt, block_size):
    pad_len = block_size - (len(txt) % block_size)
    padding = bytes([pad_len]) * pad_len
    return txt + padding
def unpad(txt, block_size):
    pad_len = txt[-1]
    if pad_len > block_size:
        raise ValueError("Padding incorrecto")
    return txt[:-pad_len]
```

Listing 7: Funciones de *padding* y *unpadding*

Para la desencriptación se sigue un patrón similar, se crearon funciones de desencriptado para cada tipo de cifrado, estas reciben el texto ya cifrado anteriormente, la *key* y su *IV* correspondientes, crea el objeto de desencriptado a través de la librería, el texto encriptado se desencripta a través de esta función, para luego eleminar el *padding* agregado antes de haber sido encriptado y finalmente se decodifica usando '*utf-8*' para hacerlo legible.

```python
def DES_decrypt(txt_encrypted, key, IV):
    deenc = DES.new(key, DES.MODE_CBC, IV)
    return unpad(deenc.decrypt(txt_encrypted), DES.block_size).decode("utf
   -8")

```

```
def DES3_decrypt(txt_encrypted, key, IV):
    deenc = DES3.new(key, DES3.MODE_CBC, IV)
    return unpad(deenc.decrypt(txt_encrypted), DES3.block_size).decode("utf-8")

def AES256_decrypt(txt_encrypted, key, IV):
    deenc = AES.new(key, AES.MODE_CBC, IV)
    return unpad(deenc.decrypt(txt_encrypted), AES.block_size).decode("utf-8")
```

Listing 8: Funciones de desencriptado

```
==========================
==== Desencriptando... ====
==========================

Texto desencriptado con DES: ola
Texto desencriptado con 3DES: ola
Texto desencriptado con AES-256: ola
```

Figura 5: Texto encriptado anteriormente desencriptado

2.5. Compara los resultados con un servicio de cifrado online

La página 'online tools' cuenta con varias herramientas bastante útiles para distintos ámbitos, a través de esta, se encontraron las siguientes:

- online-tools: DES, para encriptar utilizando *DES*.
- online-tools: triple-DES, para encriptar utilizando *3DES*.
- online-tools: AES, para encriptar utilizando *AES-256*.

Para cada cifrado te permite distintas opciones, entre las cuales las que nos interesan principalmente es el tipo de *padding*, ya que con esta opción se puede seleccionar el mismo implementado en el código. También a través de la selección de '*Key Type*' en *Custom* se puede ingresar una *Key* e *IV* propios, también el modo de encriptado (en nuestro caso *CBC*) y el resultado en hexadecimal o *base64*, aparte de obviamente el texto a encriptar.

Cómo se observa en la siguiente imagen, se encriptó el texto **'ola'** con los siguientes parámetros para cada cifrado:

1. ***DES*:**
    - ***Key*:** 'colocolo'.
    - ***IV*:** 'colocolo'.

2. ***3DES***:
   - ***Key***: 'colocolocampeonlibertado'.
   - ***IV***: 'colocolo'.
3. ***AES-256***:
   - ***Key***: 'colocolocampeondelibertadores199'.
   - ***IV***: 'colocolocampeon1'.

```
> python3 code.py
====================
===== Ingreso =====
====================

==== DES ====
Key para DES (8 bytes): colocolo
IV para DES (8 bytes): colocolo
==== 3DES ====
Key para 3DES (24 bytes): colocolocampeonlibertadores
IV para 3DES (8 bytes): colocolo
==== AES ====
Key para AES-256 (32 bytes): colocolocampeondelibertadores1991
IV para AES-256 (16 bytes): colocolocampeon1
==== Texto ====
Texto a cifrar: ola

========================
==== Procesando... ====
========================

Clave a utilizar para DES (bytes): b'colocolo'
Clave a utilizar para 3DES (bytes): b'colocolocampeonlibertado'
Clave a utilizar para AES-256 (bytes): b'colocolocampeondelibertadores199'

========================
==== Encriptando... ====
========================

Texto encriptado con DES (base64): yQHGICg70iw=
Texto encriptado con DES (HEX): c901c620283bd22c

Texto encriptado con 3DES (base64): dYj7lgtD604=
Texto encriptado con 3DES (HEX): 7588fb960b43eb4e

Texto encriptado con AES-256 (base64): qAApbqEzwq4kx01eErAkGA==
Texto encriptado con AES-256 (HEX): a800296ea133c2ae24c74d5e12b02418

============================
==== Desencriptando... ====
============================

Texto desencriptado con DES: ola
Texto desencriptado con 3DES: ola
Texto desencriptado con AES-256: ola
```

Figura 6: Output completo de código

Comparando el cifrado obtenido a través del código con el de la página (en hexadecimal, pero en *base64* el resultado sería el mismo), obtenemos lo siguiente:

1. ***DES*:**
   - **Cifrado código:** 'c901c620283bd22c'.
   - **Cifrado página:** 'c901c620283bd22c'.

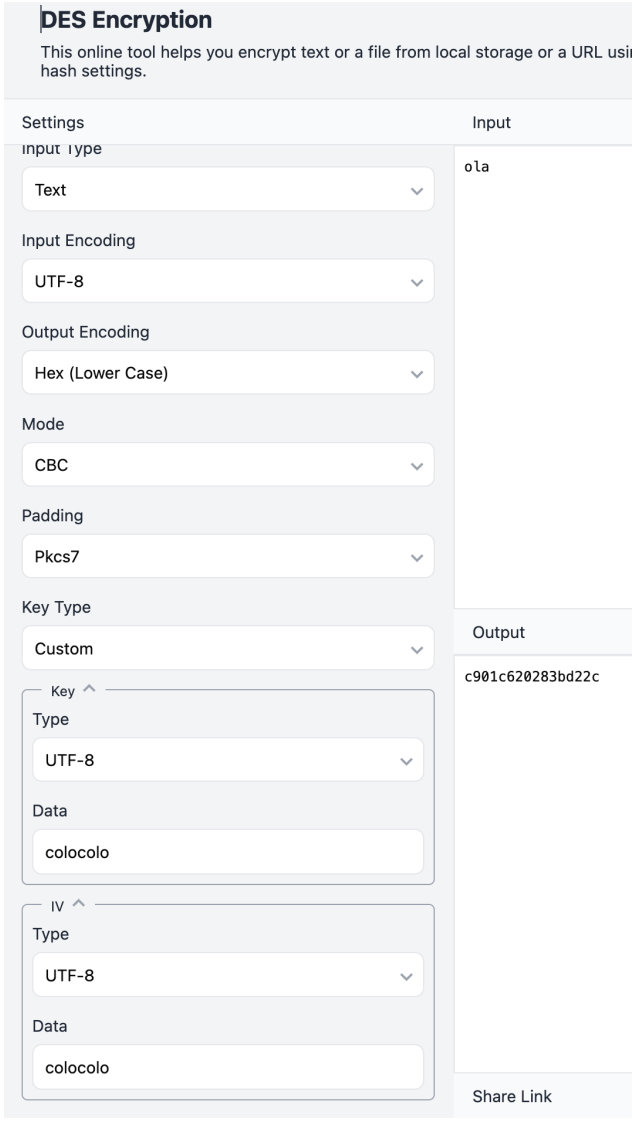

Figura 7: Cifrado *DES* a través de página

2. ***3DES*:**
   - **Cifrado código:** '7588fb960b43eb4e'.

- **Cifrado página:** ’7588fb960b43eb4e’.

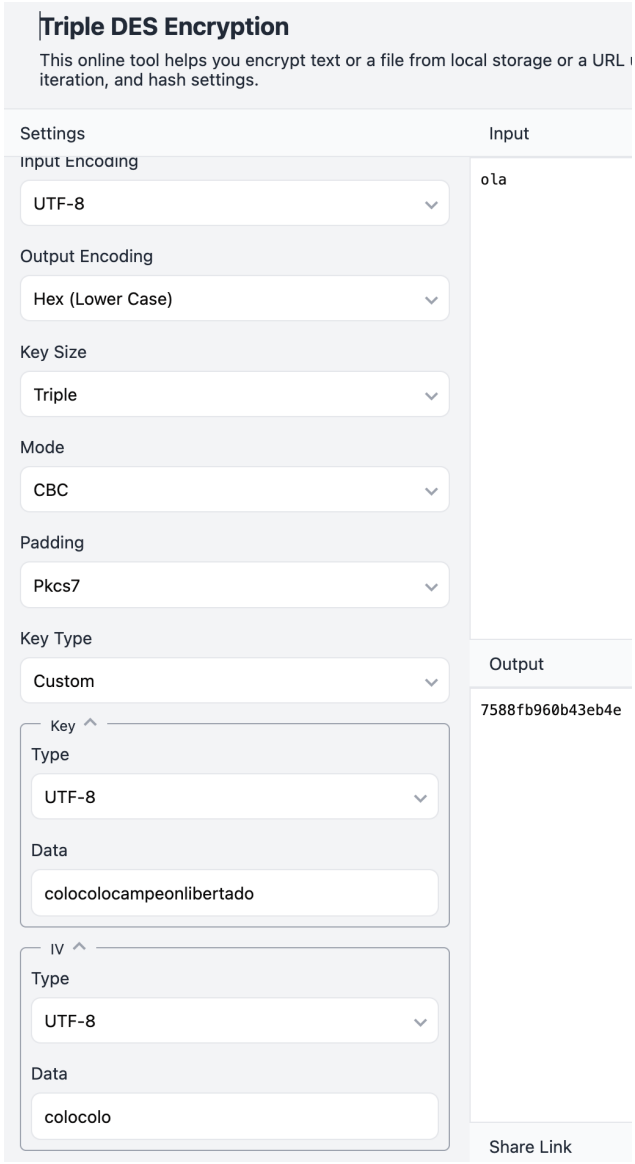

Figura 8: Cifrado *3DES* a través de página

3. ***AES-256*:**

- **Cifrado código:** ’a800296ea133c2ae24c74d5e12b02418’.
- **Cifrado página:** ’a800296ea133c2ae24c74d5e12b02418’.

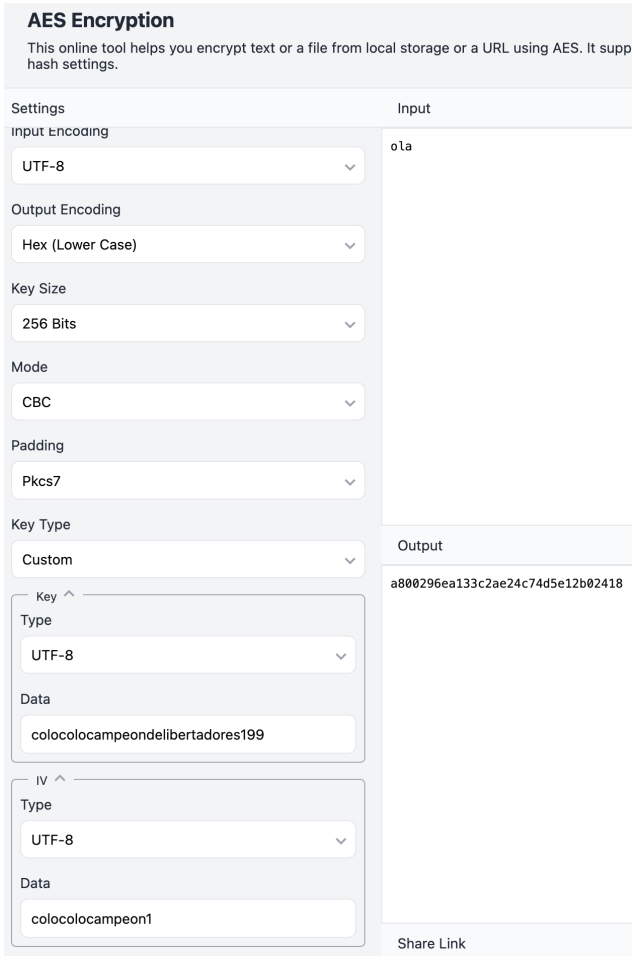

Figura 9: Cifrado *AES-256* a través de página

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

Un caso de uso algo cotidiano podría ser proteger información sensible que no se debería ver por terceros, ya sea porque es algo personal o que contiene datos importantes tuyos o alguna hipotética empresa, datos bancarios, etc. Esta información que no deben ver terceros aunque estén almacenados en una computadora de forma local podrían verse filtrados por algún malware o robo de tal computadora, y una forma de mantenerlos a salvo bajo cualquier circunstancia sería tenerlos cifrados, de esta manera solo la persona que sepa la llave podría acceder a ellos, descifrandolos cada vez que los vaya a utilizar y volviendo a cifrarlos antes de guardarlo. El algoritmo ideal sería *AES-256*, primero que nada porque *DES* sufre de muchos problemas y vulnerabilidades ante ataques de fuerza bruta, lo cual lo hace un algoritmo fácil de romper y obtener su contenido, por otro lado, *3DES* si tiene mayor seguridad pero sigue siendo algo vulnerable ante ataques de fuerza bruta y también muy lento de operar, por esto mismo estos algoritmos dejaron de ser el estándar, aquel que si es estándar y se recomienda usarlo es *AES-256*, principalmente en esta versión (con *key* de 256 bits) ya que representa

una mayor seguridad, también una mucha mayor velocidad que *3DES*. Por lo tanto el mejor algoritmo (por seguridad y velocidad) sería *AES-256*.

La sugerencia de *hash* en reemplazo de un cifrado representa un error de conceptos respecto del objetivo de cada uno de estos algoritmos, esto ya que apuntan a cosas similares pero no exactamente lo mismo, mientras que el cifrado con la *key* e *IV* correcto puede revertirse, un buen *hash* (como *SHA-256*) tiene como meta que no se pueda revertir, que con el resultado de este no sea posible llegar al *input* principal. Esta diferencia de objetivos es clave, ya que si queremos mantener algo seguro pero si poder recuperarlo en un futuro, la opción **debe** ser un cifrado idealmente muy seguro, con un *hash* podríamos dejarlo muy seguro pero no recuperarlo ni siquiera nosotros mismos.

Conclusiones y comentarios

Gracias a este laboratorio se comprendió muy bien la forma en la que trabajan los algoritmos de cifrado *DES*, *3DES* y *AES-256*, junto con la importancia de sus *key* e *IV* y el impacto que tiene esto en su seguridad.

Por otro lado, la elección de libería debería ser más cuidada, ya que '*pycrypto*' no se encuentra en versiones actuales de python, dificultando su uso para desarrollar la actividad, esta podría haberse visto reemplazada por la librería '*PyCryptodome*', la cual si se encuentra fácilmente en las versiones recientes del lenguaje y tiene mayores utilidades respecto a un posible uso cotidiano.