Informe Laboratorio 4

Sección 2

Dante Hortuvia
e-mail: dante.hortuvia@mail.udp.cl

Diciembre 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

3. Algoritmos de Cifrado: DES, 3DES y AES-256

El algoritmo **DES (Data Encryption Standard)** cifra información en bloques de 64 bits, utilizando una clave nominal de 64 bits, aunque en realidad solo 56 bits son efectivos, ya que cada octavo bit se usa para paridad. Este método se considera actualmente inseguro debido a su vulnerabilidad ante ataques de fuerza bruta.

El **3DES (Triple DES)** surge como una mejora de DES, aplicando el cifrado tres veces sobre cada bloque de 64 bits. Puede emplear dos o tres claves independientes, generando longitudes efectivas de 112 o 168 bits (128 o 192 bits totales antes de eliminar los bits de paridad). Aunque ofrece mayor seguridad, su principal desventaja es la ineficiencia y lentitud frente a algoritmos más modernos.

Por su parte, el **AES-256 (Advanced Encryption Standard)** es el estándar actual de cifrado simétrico. Opera con bloques de 128 bits y una clave de 256 bits, ofreciendo una seguridad significativamente superior y una alta eficiencia computacional. Su diseño lo hace resistente a ataques de fuerza bruta y ampliamente adoptado en sistemas modernos.

En cuanto a los **vectores de inicialización (IV)**, estos añaden aleatoriedad al proceso de cifrado, evitando la detección de patrones entre mensajes cifrados. Su tamaño depende del tamaño del bloque del algoritmo: 64 bits (8 bytes) para DES y 3DES, y 128 bits (16 bytes) para AES-256.

Tabla 1: Tamaños de clave, bloque e IV para los algoritmos DES, 3DES y AES-256.

Algoritmo	**Tamaño de clave**	**Tamaño de bloque / IV**	**Fuente principal**
DES	8 bytes	8 bytes	NIST FIPS 46-3
3DES (TDEA)	16 o 8 bytes	64 bits	NIST SP 800-67 Rev. 2
AES-256	32 bytes	16 bytes	NIST FIPS 197

DES y 3DES operan sobre bloques de 64 bits, AES-256 utiliza bloques de 128 bits, ofreciendo una mejor relación entre seguridad y rendimiento. Por esta razón, AES-256 es actualmente el estándar de cifrado más utilizado en aplicaciones que requieren alta seguridad y eficiencia.

3.1. Solicita datos de entrada desde la terminal

Se implementa un pequeño menu para decidir que tipo de algoritmo quiere usar el usuario, una vez decidido el algortimo a usar le pide el ingreso de datos mediante ïnputs", para luego proceder con ciertos ajuster para la llave y el Vector_IV

Listing 1: Pequeño menu para la utilizacion de los cifrados

```
while 1 :

    print("Ingrese tipo de Cifrado")
    print("[1] Para DES")
    print("[2] Para AES-256")
    print("[3] para 3DES")
    print("Cualquier otro valor para Salir")

    Cifrado = int(input())
    if Cifrado not in [1, 2, 3] :
        break
    Key = input("Ingrese clave a Para cifrar : \n")
    Key = Key.encode('utf-8') #Para pasar a bytes
    Valided_Key = creacion_Clave(Key,Cifrado)

    Vector = input("ingrese valor del vector IV : \n")
    Vector = Vector.encode('utf-8')

    Texto = input("ingrese texto a para cifrar : \n")
    Texto = Texto.encode('utf-8')

    if Cifrado == 1 :
        C_Text,Vector_IV = Encriptacion_DES(Valided_Key,Vector,Texto
            )
        Texto_Original = Desencriptacion_DES(Valided_Key,Vector_IV,
            C_Text)
    elif Cifrado == 3 :
        C_Text,Vector_IV = Encriptacion_3DES(Valided_Key,Vector,
            Texto)
        Texto_Original = Desencriptacion_3DES(Valided_Key,Vector_IV,
            C_Text)
    elif Cifrado == 2 :
        C_Text,Vector_IV = Encriptacion_AES_256(Valided_Key,Vector,
            Texto)
        Texto_Original = Desencriptacion_AES_256(Valided_Key,
            Vector_IV,C_Text)
```

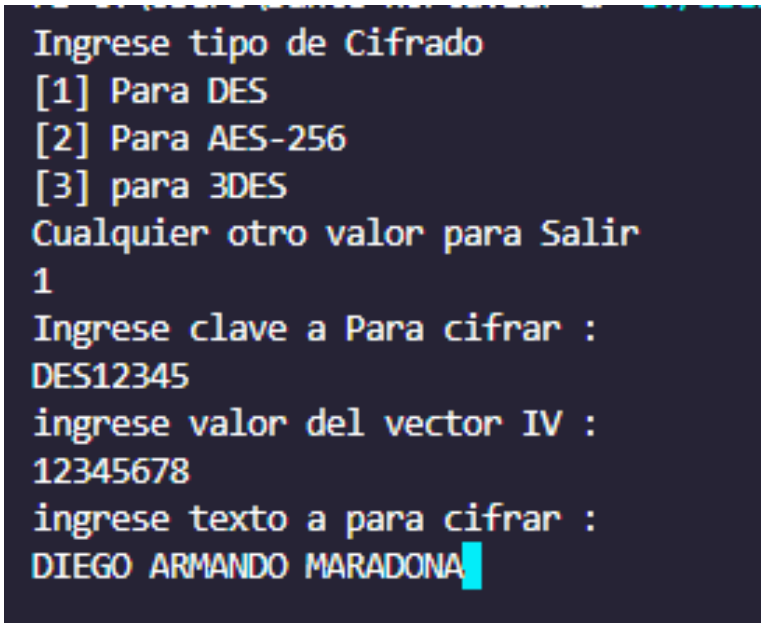

Figura 1: Ejemplo de el ingreso de los datos a cifrar.

Como se observa en la imagen anterior, los datos se ingresan mediante la función `input()`, acompañada de un `print` que indica qué dato se solicita y en qué orden deben introducirse. Es importante destacar que los datos solo pueden ingresarse en texto plano, ya que el programa no admite valores en formato hexadecimal. Esto se debe a que no se implementaron las funciones necesarias para realizar la conversión de dichos valores.

3.2. Valida y ajusta la clave según el algoritmo

Se implementa una fucion que primero verifica el tamaño de la Llave ingresada, para verificar si el tamaño de esta es correcto, esta denominada **creacion_Clave**

Listing 2: Funcion **creacion_Clave**

```
def creacion_Clave(Key, Cifrado):
    if Cifrado == 1 :
        Tama o_bytes = 8
    elif Cifrado == 2 :
        Tama o_bytes = 32
    elif Cifrado == 3 :
        Tama o_bytes = 24

    if len(Key) < Tama o_bytes :
        Key += get_random_bytes(Tama o_bytes - len(Key))
    elif len(Key) > Tama o_bytes :
        Key = Key[:Tama o_bytes]

```

```
14      return Key
```

Como se observa en el codigo anterior este ajusta la longitud de la clave según el algoritmo de cifrado seleccionado. Dependiendo del valor de Cifrado, se asigna un tamaño específico: 8 bytes para DES, 24 bytes para 3DES y 32 bytes para AES-256. Si la clave proporcionada es más corta que el tamaño requerido, la función la completa con bytes aleatorios generados mediante get_random_bytes; en caso contrario, si es más larga, se recorta hasta alcanzar la longitud adecuada. De esta forma, la función asegura que la clave cumpla con los requisitos de cada método de cifrado y sea válida para su correcto funcionamiento.

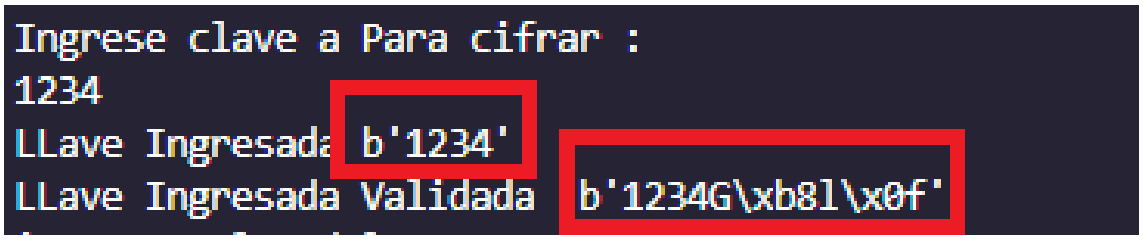

Figura 2: Ejecucion de la funcion que ajusta la llave.

Como se observa en la imagen anterior, la llave ingresa fue de 4 Bytes y la funcion le agrego (en Hexadecimal) los 4 Bytes faltantes para generar una llave de cifrado correcta.

Ademas se implementa la misma logica de funcion para el Vector_IV, donde este tambien tiene que tener una cantidad de Bytes, este tal como la funcion previamente mencionada hace el mismo proceso de trunca o añadir Bytes.

Listing 3: Variacion de la funcion para Vector_IV

```
1  if len(Vector_IV) < 8 :
2          Vector_IV += get_random_bytes(8  - len(Vector_IV))
3
4      elif len(Vector_IV) > 8 :
5          Vector_IV = Vector_IV[:8]
```

finalmente la función **creacion_Clave** garantiza que la clave utilizada cumpla con la longitud requerida por cada algoritmo de cifrado, completando o truncando los bytes según sea necesario. De esta forma, se evita que una clave inválida impida el proceso de encriptación. donde misma lógica tambien aplica al **Vector_IV**, asegurando que ambos parámetros tengan el tamaño correcto para un funcionamiento adecuado del sistema de cifrado.

3.3. Implementa el cifrado y descifrado en modo CBC

Listing 4: Funcion de encriptacion DES

```
1  def Encriptacion_DES(Key , Vector_IV, Texto) :
2
3      if len(Vector_IV) < 8 :
```

```
        Vector_IV += get_random_bytes(8  - len(Vector_IV))

    elif len(Vector_IV) > 8 :
        Vector_IV = Vector_IV[:8]

    Cifrado = DES.new(Key, DES.MODE_CBC,iv = Vector_IV)
    Texto_Con_Padding = pad(Texto,DES.block_size)
    Texto_Cifrado = Cifrado.encrypt(Texto_Con_Padding)

    return Texto_Cifrado,Vector_IV
```

Listing 5: Funcion Encriptacion 3DES

```

def Encriptacion_3DES(Key , Vector_IV, Texto) :

    if len(Vector_IV) < 8 :
        Vector_IV += get_random_bytes(8  - len(Vector_IV))

    elif len(Vector_IV) > 8 :
        Vector_IV = Vector_IV[:8]

    Cifrado = DES3.new(Key, DES3.MODE_CBC,iv = Vector_IV)
    Texto_Con_Padding = pad(Texto,DES3.block_size)
    Texto_Cifrado = Cifrado.encrypt(Texto_Con_Padding)

    return Texto_Cifrado,Vector_IV
```

Listing 6: Encriptacion AES 256

```
def Encriptacion_AES_256(Key , Vector_IV, Texto) :

    if len(Vector_IV) < 16 :
        Vector_IV += get_random_bytes(16  - len(Vector_IV))

    elif len(Vector_IV) > 16 :
        Vector_IV = Vector_IV[:16]

    Cifrado = AES.new(Key, AES.MODE_CBC,iv = Vector_IV)
    Texto_Con_Padding = pad(Texto,AES.block_size)
    Texto_Cifrado = Cifrado.encrypt(Texto_Con_Padding)
    return Texto_Cifrado,Vector_IV
```

Las funciones **Encriptacion_DES**, **Encriptacion_3DES** y **Encriptacion_AES_256** tienen como objetivo realizar el proceso de cifrado de un texto utilizando los algoritmos DES, 3DES y AES-256 respectivamente, todos bajo el modo de operación **CBC (Cipher Block Chaining)**.

Cada función comienza verificando el tamaño del **Vector de Inicialización (IV)**, el cual debe coincidir con el tamaño del bloque que utiliza cada algoritmo: 8 bytes para DES y 3DES, y 16 bytes para AES-256. Si el vector es más corto, se completa con bytes aleatorios generados mediante `get_random_bytes`; si es más largo, se trunca hasta alcanzar la longitud adecuada.

Posteriormente, se crea un objeto de cifrado con la clave (`Key`) y el vector (`Vector_IV`) mediante las librerías correspondientes (`DES.new`, `DES3.new` y `AES.new`). Antes de cifrar, el texto se rellena con la función `pad()` para asegurar que su longitud sea múltiplo del tamaño del bloque. Finalmente, se ejecuta el proceso de encriptación mediante `encrypt()`, retornando el texto cifrado junto con el vector de inicialización utilizado.

```
MESSI1234
LLave Ingresada b'MESSI1234'
LLave Ingresada Validada  b'MESSI123'
ingrese valor del vector IV :
12345678
ingrese texto a para cifrar :
EL PEPE ETE SECHHHH
Llave cifrada : b'4d45535349313233'
Llave cifrada en Hexadecimal : b'MESSI123'
Texto cifrado : b'7594ff3314521a07785279100dddd4f16142e6560f013096'
Texto cifrado en Hexadecimal : b'u\x94\xff3\x14R\x1a\x07xRy\x10\r\xdd\xd4\xf1aB\xe6V\x0f\x010\x96'
```

Figura 3: Ejecucion de cifrado DES.

```
Ingrese clave a Para cifrar :
HESOYAM
LLave Ingresada b'HESOYAM'
LLave Ingresada Validada  b'HESOYAM%\rL\xda\xcd\x1fH\x8d\xba\x7f\xfcU\x04\xa1"\xaf\xad.k\xe9\x8b\xd5q!\xe3'
ingrese valor del vector IV :
93418912374891274
ingrese texto a para cifrar :
Ah sh*t, here we go again
Llave cifrada : b'4845534f59414d250d4cdacd1f488dba7ffc5504a122afad2e6be98bd57121e3'
Llave cifrada en Hexadecimal : b'HESOYAM%\rL\xda\xcd\x1fH\x8d\xba\x7f\xfcU\x04\xa1"\xaf\xad.k\xe9\x8b\xd5q!\xe3'
Texto cifrado : b'4457222fd7f29461f14a97da08dcddf3dc2149118d2d9afa3d2edb7998ea6d63'
Texto cifrado en Hexadecimal : b'DW"/\xd7\xf2\x94a\xf1J\x97\xda\x08\xdc\xdd\xf3\xdc!I\x11\x8d-\x9a\xfa=.\xdby\x98\xeamc'
```

Figura 4: Ejecucion de cifrado AES_256.

```
Ingrese clave a Para cifrar :
Kanye West
LLave Ingresada b'Kanye West'
LLave Ingresada Validada  b'Kanye West\xaf@\x9a#]\xe99h\x1c\xa3\xff\x13\xc58'
ingrese valor del vector IV :
12431515135135
ingrese texto a para cifrar :
Kim left me, nineteen dollar Fortnite card
Llave cifrada : b'4b616e796520576573746af409a235de939681ca3ff13c538'
Llave cifrada en Hexadecimal : b'Kanye West\xaf@\x9a#]\xe99h\x1c\xa3\xff\x13\xc58'
Texto cifrado : b'3d864cc0ada67acc489c8faadd0da6331efc3e5e1f3a2647a8561452d011f6dd94c8a94b10a2395c184304e548cc0ae3'
Texto cifrado en Hexadecimal : b'=\x86L\xc0\xad\xa6z\xccH\x9c\x8f\xaa\xdd\r\xa63\x1e\xfc>^\x1f:&G\xa8V\x14R\xd0\x11\xf6\xdd\x94\xc8\xa9K\x10\xa29\\\x18C\x04\xe5H\xcc\n\xe3'
```

Figura 5: Ejecucion de cifrado 3DES.

Como se aprecia en los resultados, cada algoritmo ajusta automáticamente tanto la clave como el vector de inicialización (**IV**) según el tamaño requerido. En los casos en que la

clave ingresada es más corta, el programa completa los bytes faltantes con valores aleatorios, garantizando que la longitud cumpla con el estándar del cifrado (8 bytes para DES, 24 bytes para 3DES y 32 bytes para AES-256).

Durante la ejecución se muestra la clave original, la clave validada (ya ajustada) y el texto cifrado tanto en formato hexadecimal como en texto plano. Esto permite visualizar cómo los datos ingresados se transforman en una secuencia codificada ilegible, producto del proceso de encriptación.

En el caso del cifrado AES-256, puede observarse que la clave fue considerablemente extendida mediante bytes aleatorios, mientras que para DES y 3DES el relleno fue menor debido a su menor tamaño de bloque. En todos los casos, el algoritmo aplicó correctamente el modo **CBC**, generando resultados distintos incluso para textos similares, gracias al uso del IV y al padding aplicado antes del cifrado.

Listing 7: Desencriptacion DES

```
def Desencriptacion_DES(Key, Vector_IV, Texto_Cifrado):
    descifrado = DES.new(Key, DES.MODE_CBC, iv=Vector_IV)
    texto_plano = unpad(descifrado.decrypt(Texto_Cifrado), DES.
       block_size)
    return texto_plano
```

Listing 8: Desencriptacion 3DES

```
def Desencriptacion_3DES(Key, Vector_IV, Texto_Cifrado):
    descifrado = DES3.new(Key, DES3.MODE_CBC, iv=Vector_IV)
    texto_plano = unpad(descifrado.decrypt(Texto_Cifrado), DES3.
       block_size)
    return texto_plano
```

Listing 9: Desencriptacion AES 256

```
def Desencriptacion_AES_256(Key, Vector_IV, Texto_Cifrado):
    descifrado = AES.new(Key, AES.MODE_CBC, iv=Vector_IV)
    texto_plano = unpad(descifrado.decrypt(Texto_Cifrado), AES.
       block_size)
    return texto_plano
```

se implementa el proceso inverso al cifrado, es decir, la **desencriptación** de los datos previamente encriptados mediante los algoritmos DES, 3DES y AES-256, todos utilizando el modo de operación **CBC**.

Cada función recibe como parámetros la clave, el vector de inicialización (**IV**) y el texto cifrado. A partir de estos, se crea un objeto de descifrado con la función correspondiente (`DES.new`, `DES3.new` o `AES.new`), empleando el mismo modo y los mismos valores utilizados durante el proceso de encriptación. Luego, el texto cifrado se descifra mediante `decrypt()` y se eliminan los bytes de relleno añadidos en el proceso original con la función `unpad()`, obteniendo así el texto plano original. El correcto funcionamiento de estas funciones depende

de que la clave y el vector de inicialización coincidan exactamente con los usados durante el cifrado.

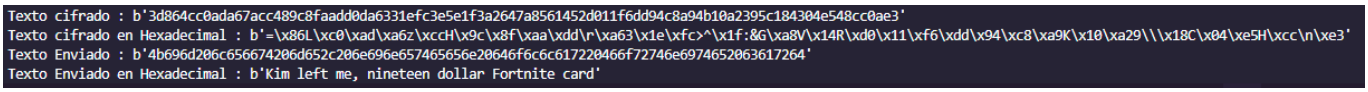

Figura 6: Ejecucion de Decifrado DES.

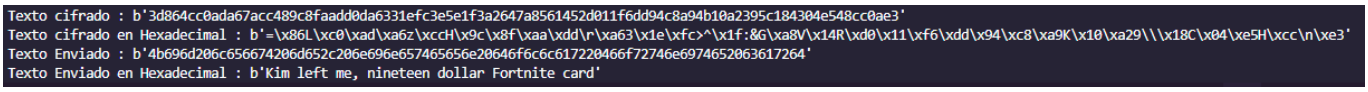

Figura 7: Ejecucion de Decifrado 3DES.

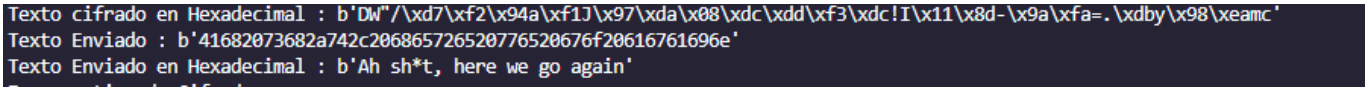

Figura 8: Ejecucion de Decifrado AES.

En los resultados de desencriptación se observa que los tres algoritmos DES, 3DES y AES-256 lograron recuperar correctamente el texto original a partir del texto cifrado, siempre que se utilizen las mismas claves y vectores de inicialización empleados durante el cifrado. Esto demuestra que las funciones implementadas realizan el proceso de descifrado de manera correcta, eliminando el padding agregado previamente y validando la integridad del mensaje original.

3.4. Compara los resultados con un servicio de cifrado online

Para hacer una comparacion de los resultados se ulitiza la siguiente web Online Tools, donde se utilizo especificamente el cifrado AES-256, esta herramienta se configuro para usar el modo CBC esta tiene distintos parámetros que permiten observar de forma práctica cómo funciona este algoritmo. El sitio solicita ingresar el texto que se desea cifrar, la clave (Key) y el vector de inicialización (IV), los cuales pueden definirse manualmente o generarse a partir de una contraseña mediante funciones de derivación como PBKDF2. Además, permite seleccionar el modo de operación (en este caso CBC), el tipo de codificación del texto (UTF-8, Hex, Base64, entre otros) y el esquema de padding que ajusta el tamaño de los bloques. Esta flexibilidad facilita comprender cómo la clave y el IV influyen en el resultado final del cifrado.

Este se configura de esta manera :

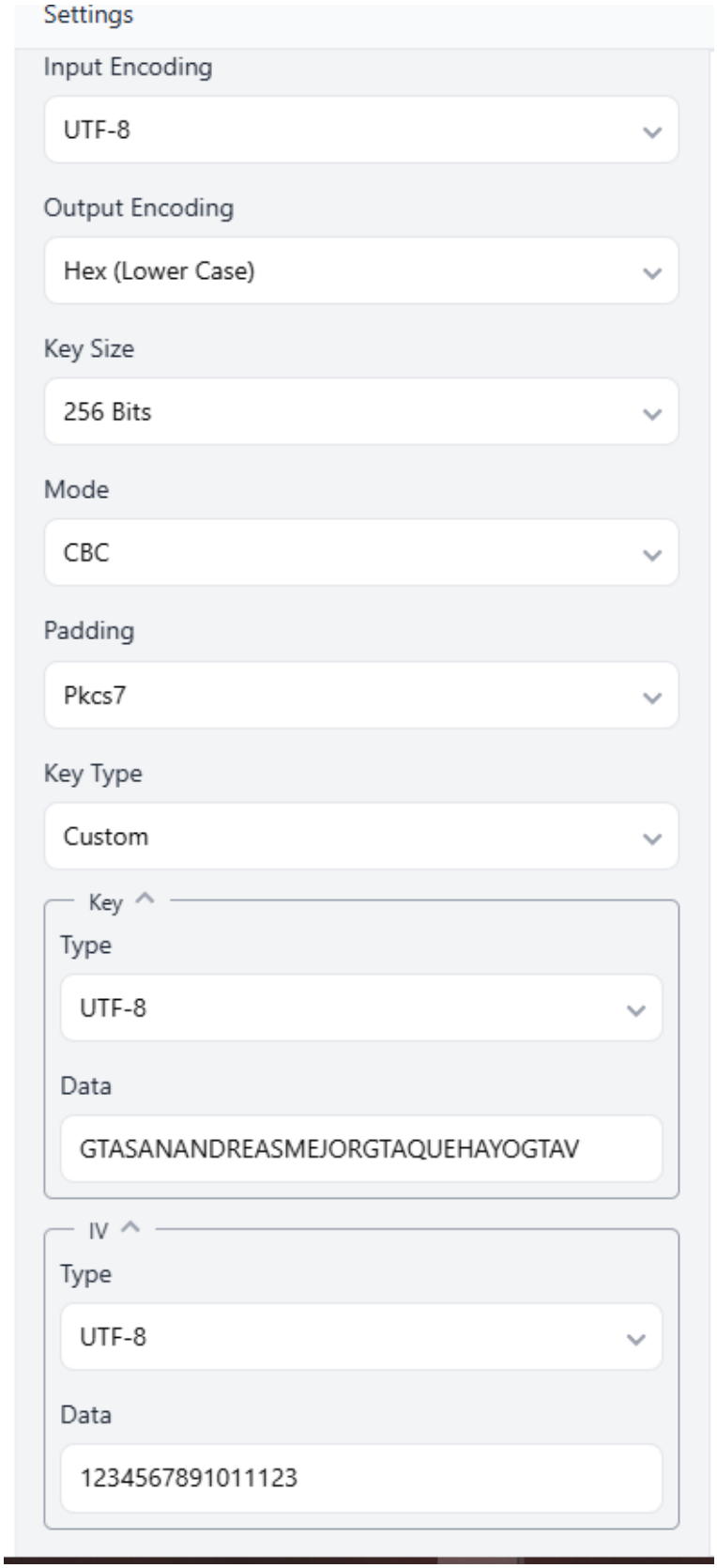

Figura 9: Datos ingresados a la web.

AES-256:

- **Key:** `GTASANANDREASMEJORGTAQUEHAYOGTAV`
- **IV:** `1234567891011123`
- **Input:** `All we had to do, was follow the damn train, CJ!`

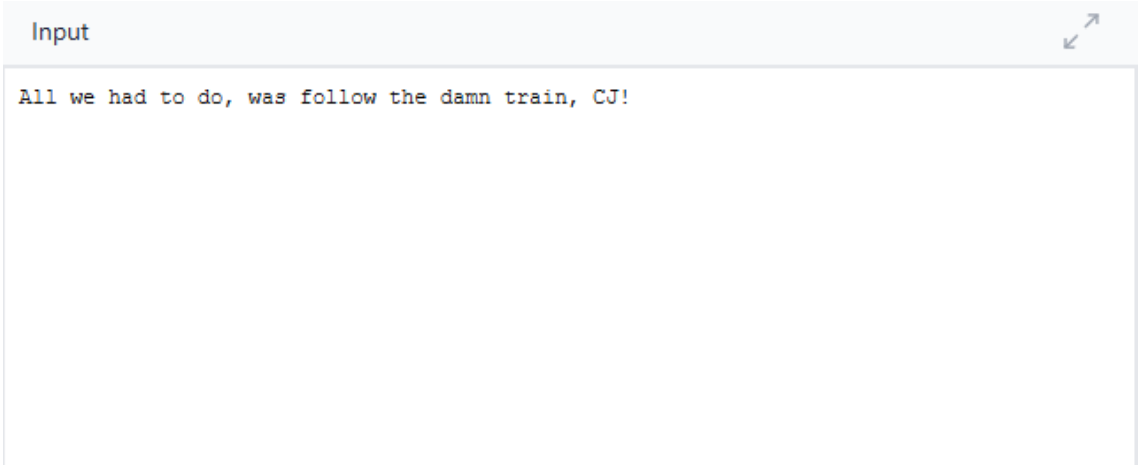

Figura 10: Input Ingresado en la web.

Donde se obtuvo los siguientes resultados :

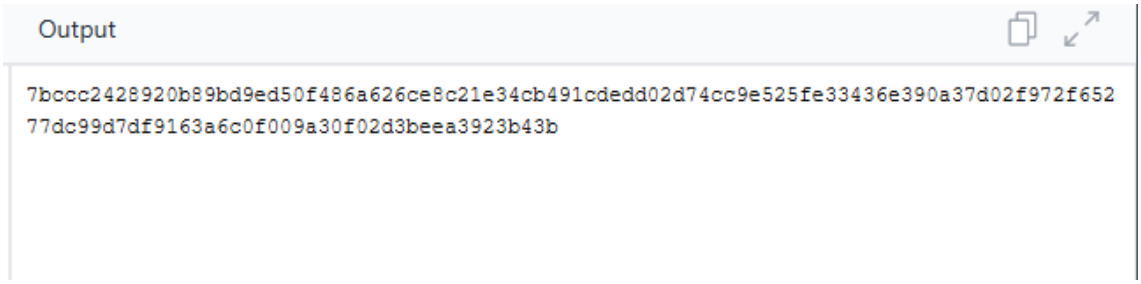

Figura 11: Resultados obtenidos de la web.

Procedemos a comprarlos con los resultados obtenidos por nuestro codigo :

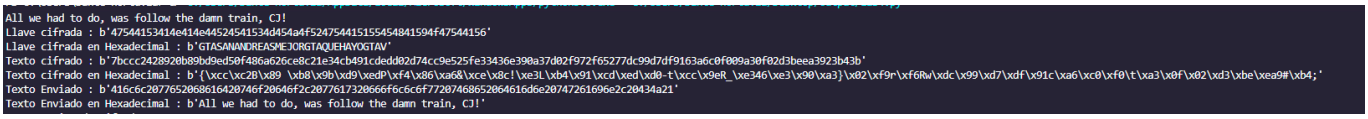

Figura 12: Resultados Obtenidos del codigo.

Como se observa en las imagenes los resultados son iguales, esto nos indica que las funciones implementadas fueron creadas de manera correcta, importante recalcar que esto funciona igual ya que se implementan la totalidad de bytes en ambos programas, si esto no fuera asi se rellenarian con bytes ramdom haciendo muy complejo que lleguen a los mismos resultados al momento de encriptar aunque mantendria el texto encriptado igual en ambos casos.

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

En la vida real, los algoritmos de cifrado simétrico como DES, 3DES y AES son ampliamente utilizados en una gran variedad de aplicaciones. Este tipo de cifrado se emplea para proteger archivos almacenados en dispositivos o servidores, en sistemas de autenticación y en la transmisión segura de datos a través de redes. Protocolos de comunicación como HTTPS, SSL/TLS y VPN utilizan cifrado simétrico para garantizar la confidencialidad e integridad de la información una vez que se establece un canal seguro.

El algoritmo AES, en particular, es actualmente el estándar más adoptado a nivel mundial, implementado en servicios como la banca en línea, aplicaciones móviles, discos cifrados y dispositivos IoT. Su gran rapidez y resistencia frente a ataques lo convierten en una herramienta esencial para la protección de la información, tanto en entornos personales como corporativos. Sin embargo, uno de los principales desafíos del cifrado simétrico es la gestión segura de las claves, ya que la misma clave debe ser compartida entre las partes. Por ello, suele combinarse con mecanismos de cifrado asimétrico para intercambiar las claves de forma segura.

Un caso práctico de uso sería proteger información sensible que no debería ser vista por terceros, como datos personales, bancarios o empresariales. Aunque dicha información esté almacenada localmente, siempre existe el riesgo de robo o filtración por malware. Para mitigar esto, la mejor opción es mantener los datos cifrados, de modo que solo quien posea la clave pueda acceder a ellos. Así, los archivos pueden descifrarse solo al momento de usarse y volver a cifrarse antes de guardarse.

Entre los algoritmos comparados, AES-256 se destaca como el más seguro y eficiente. Mientras DES presenta vulnerabilidades ante ataques de fuerza bruta y 3DES, aunque más robusto, resulta lento y también vulnerable, AES-256 ofrece un excelente equilibrio entre seguridad y rendimiento, razón por la cual se ha convertido en el estándar recomendado.

Finalmente, es importante distinguir entre cifrado y hash. Aunque ambos se relacionan con la seguridad de los datos, sus objetivos son diferentes: el cifrado permite revertir el proceso para recuperar la información original si se dispone de la clave adecuada, mientras que un hash como SHA-256 es irreversible, impidiendo obtener los datos originales desde su resultado. Por ello, si se desea proteger información que deba ser recuperada posteriormente, debe utilizarse un cifrado seguro y no un algoritmo de hash.

Conclusiones y comentarios

Gracias a este laboratorio se logró comprender en profundidad el funcionamiento de los algoritmos de cifrado simétrico DES, 3DES y AES-256, así como la relevancia que tienen las claves (Key) y los vectores de inicialización (IV) en la seguridad del proceso de cifrado. Se implementaron correctamente los tres métodos utilizando la librería PyCryptodome en Python, la cual resultó ser una alternativa moderna y funcional frente a la obsoleta PyCrypto.

Durante la práctica se observó cómo la validación y ajuste automático de claves e IVs permite mantener la compatibilidad con los estándares del NIST, evitando errores en la ejecución. Los resultados obtenidos coincidieron con los de herramientas de cifrado en línea, confirmando la correcta implementación del modo CBC y del proceso de relleno (padding).

Además, se evidenció que el uso de bytes aleatorios para completar claves o IVs influye directamente en el resultado del cifrado, lo que resalta la importancia de usar claves de longitud adecuada y gestionarlas de manera segura. En términos prácticos, el laboratorio permitió apreciar la utilidad del cifrado simétrico en la protección de información sensible tanto en almacenamiento como en transmisión de datos.

Entre los algoritmos estudiados, AES-256 se destacó por su mayor seguridad y eficiencia, siendo el estándar actual en sistemas como la banca en línea, almacenamiento cifrado y redes privadas. Finalmente, se concluye que la gestión segura de claves es un aspecto crítico del cifrado simétrico; por ello, en entornos reales, se recomienda complementarlo con técnicas de cifrado asimétrico o mecanismos seguros de intercambio de claves para garantizar una protección integral de la información.