SPS- TECH IS NOW

Práctica Microservicios

Documento técnico.

Micrservicios SPS: Cifrador de imagenes con AES y diversos modos de operación

Presenta:Cruz Flores Omar

omar1cruz2714721@gmail.com

Junio 30, 2020

Indice

Lista de figuras

Listings

1 Introducción

A lo largo de la historia la necesidad de ocultar la información ha ido en aumento. En la actualidad todos los dispositivos informáticos que contieneninformacióon privada cuentan con sistemas de encriptación tanto para el almacenamiento de información como para el intercambio de información entre estos sistemas informáticos.

Advanced Encryption Standard (AES) es un estandar de cifrado por bloques, y este cifrado por bloques se conoce como unidad cifradora, y dentro de ella se pueden utilizar diversos modos de operación para poder llegar a un cifrado; en esta práctica utilizo los siguientes:

- Modo Electronic codebook (ECB)
- Modo Cipher-block chaining (CBC)
- Modo Output feedback (OFB)

Para realizar esta práctica se hace uso de la herrmaienta **flask**, se emplea **python** como lenguaje para el desarrollo de los microservicios. En este documento se explica el funcionamiento general de la práctica de microservicios, con el fin de mostrar los conocimientos técnicos que poseo. Sin más preambulos se invita al lector a continuar con la lectura.

2 Desarrollo

Después de definir los puntos básicos para comprender el proposito deldesarrollo de nuestra practica *(gns3)*, como aclaré en la introducción de este trabajo, veremos una topología de red dada previamente por las especificaciones prácticas y del problema.

Preámbulo: La URL base **http://127.0.0.1:5000/api/sps/helloworld/v1** es el prefijo para todas las demas URL que se definen. Para el desarrollo de esta práctica de microservicos se realizó una versión gráfica y una versión que se consume a través de *Postman*. Abordemos primero la versión que se consume con postman.

2.1 Implementación no gráfica

Veamos la primera ruta de la API, **/AES**, esta se consume de la siguiente manera:

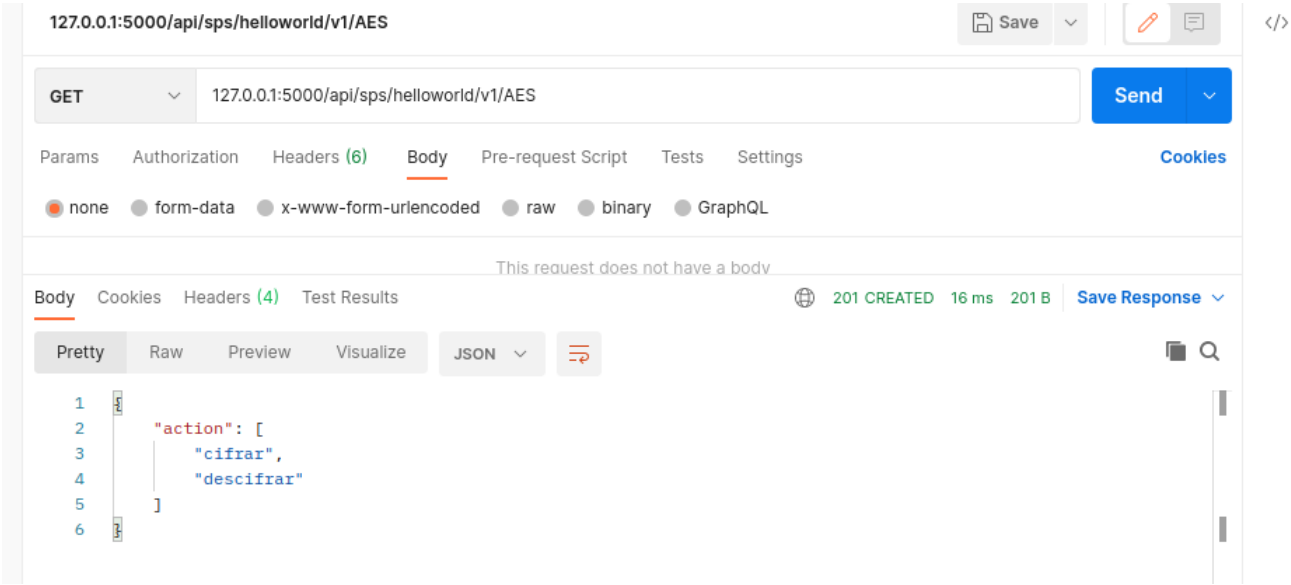

Figura 1: Postman enviando petición

El código que lo hace posible es el que se muestra

```
@app.route(base_url+"/AES")
def hello_networkers():
    return jsonify({"action": ["cifrar", "descifrar"]}),201
```

Código 1: función de la ruta principal

Como se aprecia se regresa un elemento JSON con las acciones disponiblen en el cifrador AES, así como un status 201, por default si flask no encuentra una ruta arroha un estatus 404.

Ahora pasamos a la ruta **/AES/cifrar**, la cual se consume de la siguiente manera:

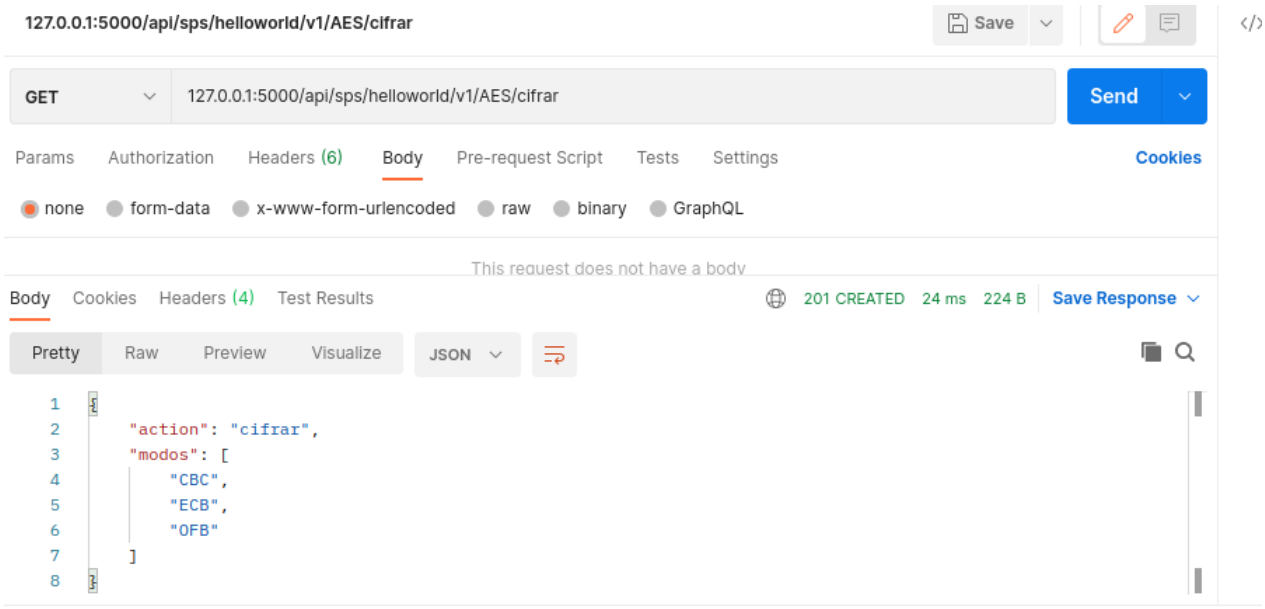

Figura 2: base_url/AES/<action>

El código que lo hace posible es el que se muestra

```
@app.route(base_url+'/AES/<action>')
def inicio(action):
    if action!= "cifrar" and action !="descifrar":
        return "bad request", 403
    return jsonify({"action":action , "modos": ["CBC", "ECB", "OFB" ]}),201
```

Código 2: función de ruta /AES/<action>

En esta ruta analizamos el parametro *<action>*, el cual solo puede caer en dos posibles datos, "cifrar" y "descifrar", si el parametro <action> es correcto se regresan todos los modos de operación que puede utilizar la API para cifrar imagenes. En caso que el parametro <action> sea diferente a "cifrar" y diferente a "descifrar" se arroja un mensaje de error y un codigo de status 403.

Hasta ahora hemos avanzado en rutas para ir recopilando información de lo que el usuario desea realizar, en este endpoint ya se tiene definido qué es lo que el usuario desea realizar (e.g el usuario desea cifrar en el modo CBC), ahora en este endpoint que ya define qué es lo que se desea hacer se puede emplear como recopilador de datos, esta ruta se consume de la siguiente manera:

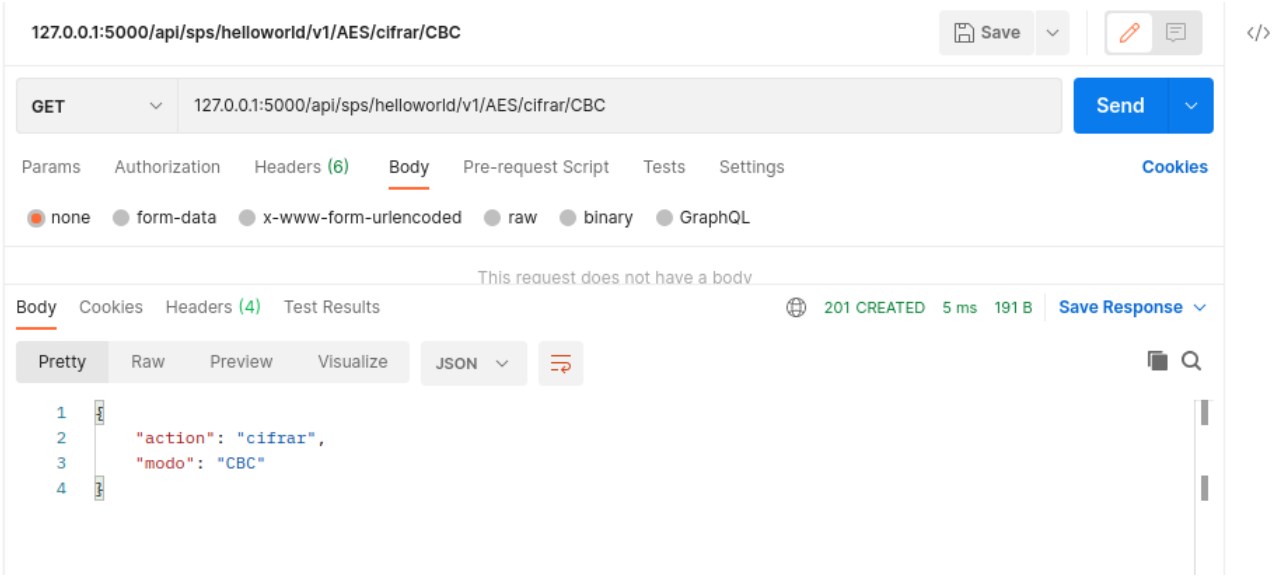

Figura 3: base_url/AES/<action>/<mode>

El código que lo hace posible es el que se muestra

```
@app.route(base_url+'/AES/<action>/<mode>')
def modo(mode,action):
    if action!= "cifrar" and action !="descifrar":
        return "bad request", 403
    if mode!= "CBC" and mode !="ECB" and mode !="OFB":
        return "bad request", 405
    return jsonify({'modo': mode, 'action': action}), 201
```

Código 3: función de ruta /AES/<action>/<mode>

En esta ruta analizamos el parametro *<action>* y *<mode>*, el primero puede caer en dos posibles datos, "cifrar" y "descifrar", mientras el segundo puede ser cualquier modo de operación que definimos en la API, en este caso son CBC, ECB y OFB si ambos parametros son correctos se regresa la acción final y el modo de operación en un JSON. En caso que el parametro <action> o <mode> no sean los esperados se arroja un mensaje de error

Ahora así que solo queda seleccionar los elementos que van a hacer posibles realizar el cifrado, y esto depende del modo de operación que se haya seleccionado, *la llave* y el *Vector de inicialización* son elementos de entrada que se colocan en este endpoint, así como el archivo que se desea cifrar. La ruta **/AES/<action>/<mode>** para este ejemplo se consume así con método POST: El código encargado de esta ruta es el siguiente

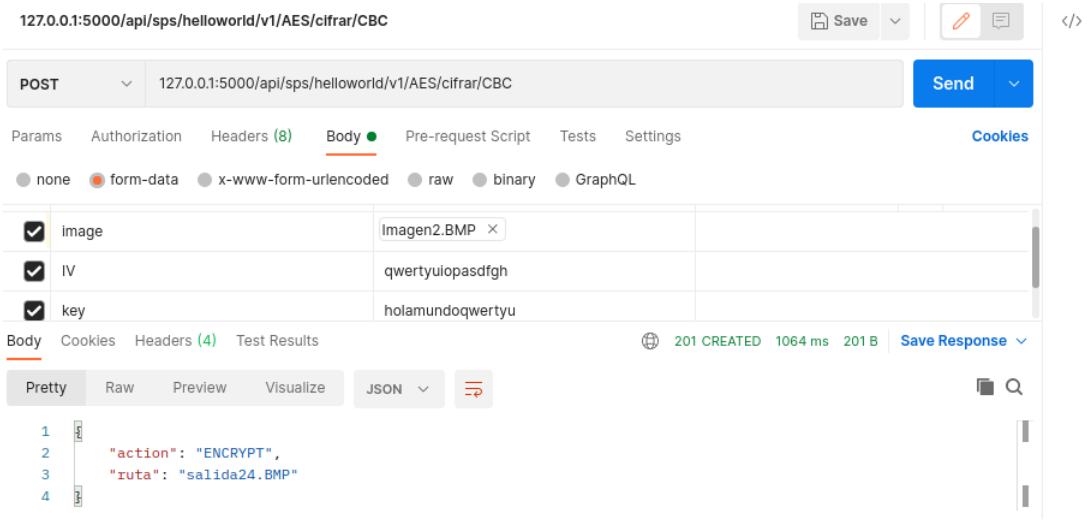

Figura 4: /AES/<action>/<mode> POST

```python
@app.route(base_url+'/AES/<action>/<mode>', methods=["POST"])
def cifrar_descifrar(mode, action):
    img=request.files["image"]
    print(img.name)
    img=Image.open(img)
    global key
    key=request.form.get("key")
    global IV
    IV=request.form.get("IV")
    if len(IV)!=16 or len(key)!=16:  #compruebo que la llave y/o el vector sean de
        return "Ocurrió un error con la longitud de llave o vector",403
    if action=="cifrar":
        if mode=="CBC":
            num=encrypt_image_cbc(img)
        elif mode=="ECB":
            num=encrypt_image_ecb(img)
        elif mode=="OFB":
            num=encrypt_image_ofb(img)
        else:
            return "NO se reconoce el modo de operación AES que deseas ejecutar",
    elif action=="descifrar":
        if mode=="CBC":
            num=AES_cbc_decrypt(img)
        elif mode=="ECB":
            num=AES_ecb_decrypt(img)
        elif mode=="OFB":
            num=AES_ofb_decrypt(img)
        else:
            return "NO se reconoce el modo de operación AES que deseas ejecutar",
    else:
```

```
31          return "NO se reconoce la acción que deseas ejecutar, asegurate que sea ci
32
33      nombre="salida"+num+"."+format
34      return jsonify({"action": action, "ruta": nombre}),201
```

Código 4: función de la ruta /AES/<action>/<mode> con POST

Como se observa se reciben de parametros *<action>* y *<mode>*, con estos parametros pasamos a dos bloques condicionales principales si el parámetro <action> desea cifrar se entra a un bloque, si se desea descifrar se entra a otro, y si no corresponde a ninguna de estas acciones se regresa un error, si se entra a un bloque de cifrado o descifrado entramos a otro bloque condicional dependiendo el **modo de operacion**, se manda a una función corrspondiente (se muestran en el código 5), y estas funciones regresan un numero, que corresponde al número de salida que se guardó la nueva imagen ya sea cifrada o descifrada en el path **API_AES/Static**. A continuación en la figura 5 se muesta la imagen original y en la figura 6 su resultado cifrado.

Por último probemos la ruta **/AES/<action>/<mode>** con la acción "descifrar", lo con-

Figura 5: Imagen original

sumimos como se muestra en la figura 7:

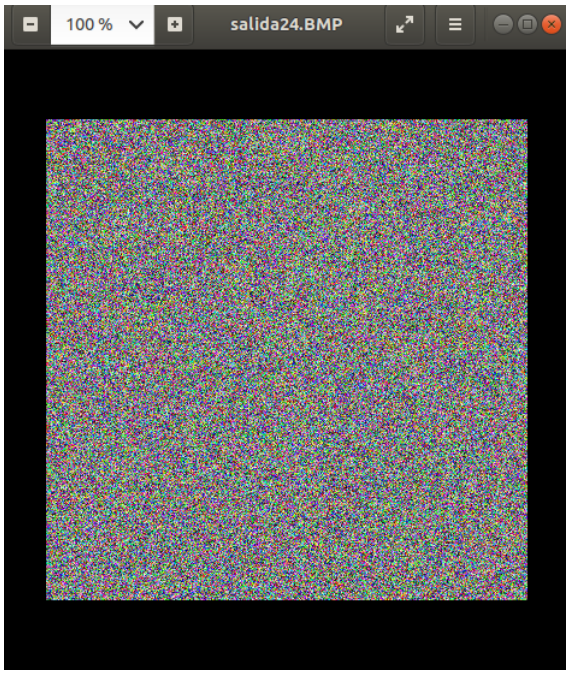

Figura 6: Imagen cifrada

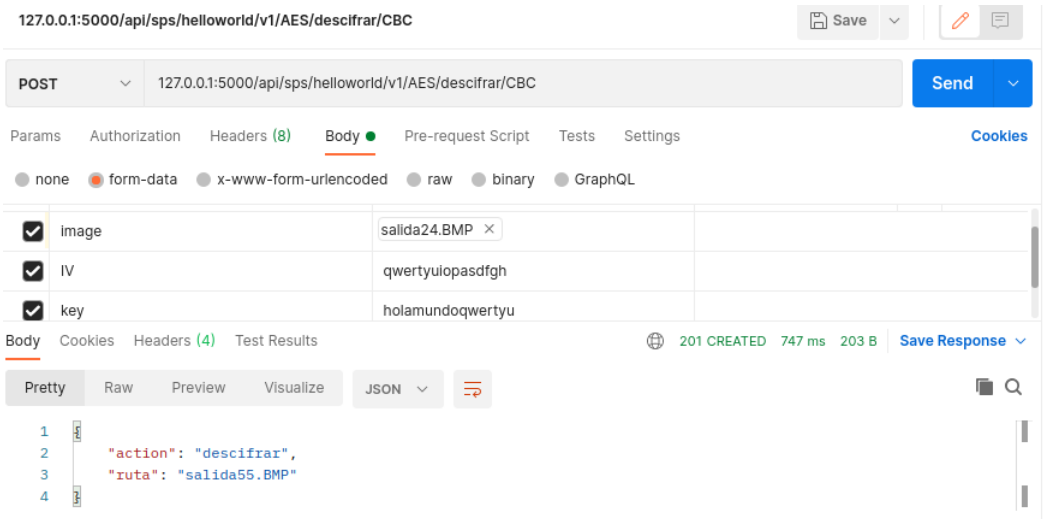

Figura 7: descifrar CBC

Como se observa colocamos la imagen que hemos cifrado, los valores del vector de inicialización y la llave (ambas de 16 bits) y devuelve un json con el nombre de la imagen resultante y la acción que se realizó. En la figura 8 se muestra la imagen resultante

Figura 8: Imagen descifrada

Las funciones que hacen posible el cifrado y descifrado se muestran a continuación, explicadas con comentarios. Es necesario recalclar que parte del código es tomada de https://www.programmerso

```python
def pad(data): #Esta función se encarga de recortar datos a 16 bits
    return data + b"\x00" * (16 - len(data) % 16)

# Mapeamos la imagen  RGB
def trans_format_RGB(data):
    #tuple: Immutable, ensure that data is not lost
    red, green, blue = tuple(map(lambda e: [data[i] for i in range(0, len(data)) if
    pixels = tuple(zip(red, green, blue))
    return pixels

########################################################  INICIO ECB ##############################
def encrypt_image_ecb(im):
    #Convertimos la imagen a pixeles RGB y despues a bytes
    value_vector = im.convert("RGB").tobytes()

    imlength = len(value_vector)

    #tranformamos a pixeles el resultado de cifrar por el metodo
    value_encrypt = trans_format_RGB(aes_ecb_encrypt(key, pad(value_vector))[:imler
    #Creamos un nuevo objeto imagen, con las dimensiones de la imagen original
    im2 = Image.new(im.mode, im.size)
    im2.putdata(value_encrypt) #colocamos los datos cifrados en los datos de la nu

    # Guardamos la imagen con el mortato que elegimos en la ruta static
    im2.save(filename_encrypted_ecb + "." + format, format)
    ruta= "/home/omar/Escritorio/SPS/API_AES/static/salida"
```

```
27      rannum=str(random.randint(0,99)) #creamos un numero aleatorio para guardar con
28      ruta+=rannum+"."+format
29      im2.save(ruta)
30      return rannum
31
32
33  def AES_ecb_decrypt(img):
34
35      #img=Image.open(filename_encrypted_ecb+"."+format)
36      value_vector = img.convert("RGB").tobytes()
37
38      imlength = len(value_vector)
39      value_encrypt=img.getdata()
40
41      ecb_decipher = AES.new(key, AES.MODE_ECB)
42      plain_data = ecb_decipher.decrypt(value_vector)
43
44      im2 = Image.new(img.mode, img.size)#referencia a la clase Image(PIL)
45      im2.putdata(trans_format_RGB(plain_data))
46      #bytes = readimage(path+extension)
47      im2.save("image_eECB_dECB.jpg")
48      ruta= "/home/omar/Escritorio/SPS/API_AES/static/salida"
49      rannum=str(random.randint(0,99))
50      ruta+=rannum+"."+format
51      im2.save(ruta)
52      return rannum
53
54  # ECB encryption
55  def aes_ecb_encrypt(key, data, mode=AES.MODE_ECB):
56      #The default mode is ECB encryption
57      aes = AES.new(key, mode)
58      new_data = aes.encrypt(data)
59      return new_data
60  ##########################################################################################fin ECB
61
62
63  ########################################################################################## CBC######
64  def encrypt_image_cbc(im):
65      #Open the bmp picture and convert it to RGB image
66      #im = Image.open(filename)
67      value_vector = im.convert("RGB").tobytes()
68
69      # Convert image data to pixel value bytes
70      imlength = len(value_vector)
71
72      # Perform pixel value mapping on the filled and encrypted data
73      value_encrypt = trans_format_RGB(aes_cbc_encrypt(key, pad(value_vector))[:imlen
74
75      # Create a new object, store the corresponding value
76      im2 = Image.new(im.mode, im.size)
77      im2.putdata(value_encrypt)
78
79      # Save the object as an image in the corresponding format
```

```
80      im2.save(filename_encrypted_cbc + "." + format, format)
81      ruta= "/home/omar/Escritorio/SPS/API_AES/static/salida"
82      rannum=str(random.randint(0,99))
83      ruta+=rannum+"."+format
84      im2.save(ruta)
85      return rannum
86      #AES_cbc_decrypt(key,IV)
87      #AES_ecb_decrypt(key)
88  def AES_cbc_decrypt(img):
89
90      #img=Image.open(filename_encrypted_cbc+"."+format)
91      value_vector = img.convert("RGB").tobytes()
92
93      imlength = len(value_vector)
94      value_encrypt=img.getdata()
95
96      cfb_decipher = AES.new(key, AES.MODE_CBC, IV)
97      plain_data = cfb_decipher.decrypt(value_vector)
98      im2 = Image.new(img.mode, img.size)#referencia a la clase Image(PIL)
99      im2.putdata(trans_format_RGB(plain_data))
100     #bytes = readimage(path+extension)
101     im2.save("image_eCBC_dCBC.jpg")
102     ruta= "/home/omar/Escritorio/SPS/API_AES/static/salida"
103     rannum=str(random.randint(0,99))
104     ruta+=rannum+"."+format
105     im2.save(ruta)
106     return rannum
107
108
109
110
111 # CBC encryption
112 def aes_cbc_encrypt(key, data, mode=AES.MODE_CBC):
113     #IV is a random value
114     #global IV
115     #IV = key_generator(16)
116     print("           ",IV)
117     aes = AES.new(key, mode, IV)
118     new_data = aes.encrypt(data)
119
120     return new_data
121 ############################################################################## FIN CBC ######
122
123 ############################################################################## OFB######
124 def encrypt_image_ofb(im):
125     #Open the bmp picture and convert it to RGB image
126     #im = Image.open(filename)
127     value_vector = im.convert("RGB").tobytes()
128
129     # Convert image data to pixel value bytes
130     imlength = len(value_vector)
131
132     # Perform pixel value mapping on the filled and encrypted data
```

```
133     value_encrypt = trans_format_RGB(aes_ofb_encrypt(key, pad(value_vector))[:imler
134
135     # Create a new object, store the corresponding value
136     im2 = Image.new(im.mode, im.size)
137     im2.putdata(value_encrypt)
138
139     # Save the object as an image in the corresponding format
140     im2.save(filename_encrypted_ofb + "." + format, format)
141     ruta= "/home/omar/Escritorio/SPS/API_AES/static/salida"
142     rannum=str(random.randint(0,99))
143     ruta+=rannum+"."+format
144     im2.save(ruta)
145     return rannum
146
147 def AES_ofb_decrypt(img):
148
149     #img=Image.open(filename_encrypted_ofb+"."+format)
150     value_vector = img.convert("RGB").tobytes()
151
152     imlength = len(value_vector)
153     value_encrypt=img.getdata()
154
155     cfb_decipher = AES.new(key, AES.MODE_OFB, IV)
156     plain_data = cfb_decipher.decrypt(value_vector)
157     im2 = Image.new(img.mode, img.size)#referencia a la clase Image(PIL)
158     im2.putdata(trans_format_RGB(plain_data))
159     #bytes = readimage(path+extension)
160     im2.save("image_eOFB_dOFB.jpg")
161     ruta= "/home/omar/Escritorio/SPS/API_AES/static/salida"
162     rannum=str(random.randint(0,99))
163     ruta+=rannum+"."+format
164     im2.save(ruta)
165     return rannum
166
167 # OFB encryption
168 def aes_ofb_encrypt(key, data, mode=AES.MODE_OFB):
169     #IV is a random value
170     #global IV
171     #IV = key_generator(16)
172     print("           ",IV)
173     aes = AES.new(key, mode, IV)
174     new_data = aes.encrypt(data)
175
176     return new_data
177 ############################################################################ FIN OFB #####
```

Código 5: programa principal

2.2 implementación gráfica

Los elementos que permiten la ejecución lógica son los mismos que en la sección 2.1, pero ahora con el uso de templates consumo la API de una manera gráfica como muestro a continuación en la figura 9-14. En la ruta **/AES**

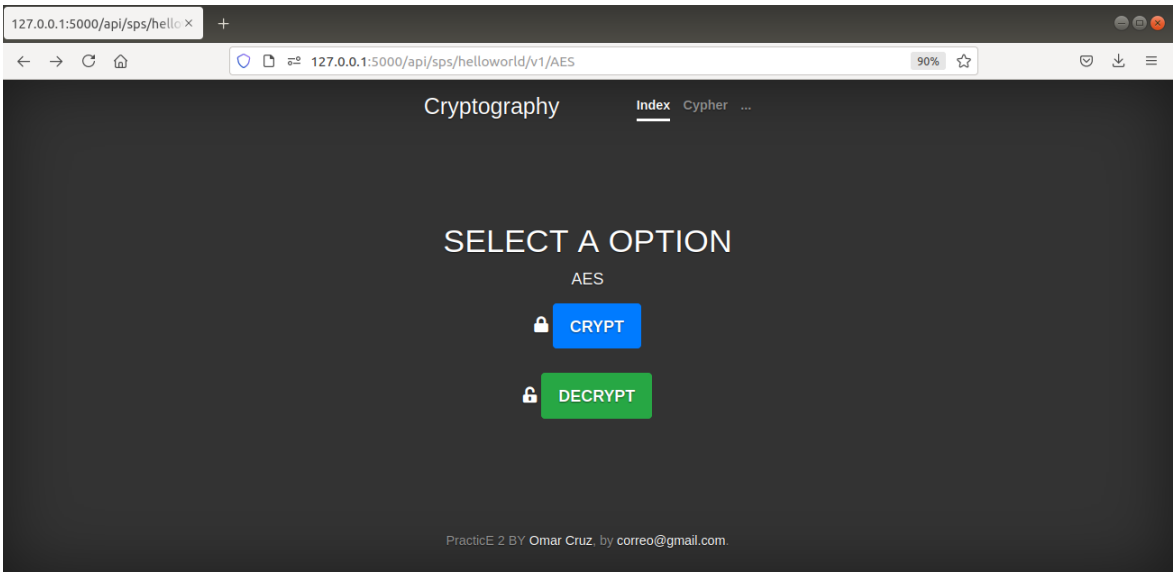

Figura 9: inicio

En la ruta **/AES/{action}/**: con el parametro action= cifrar

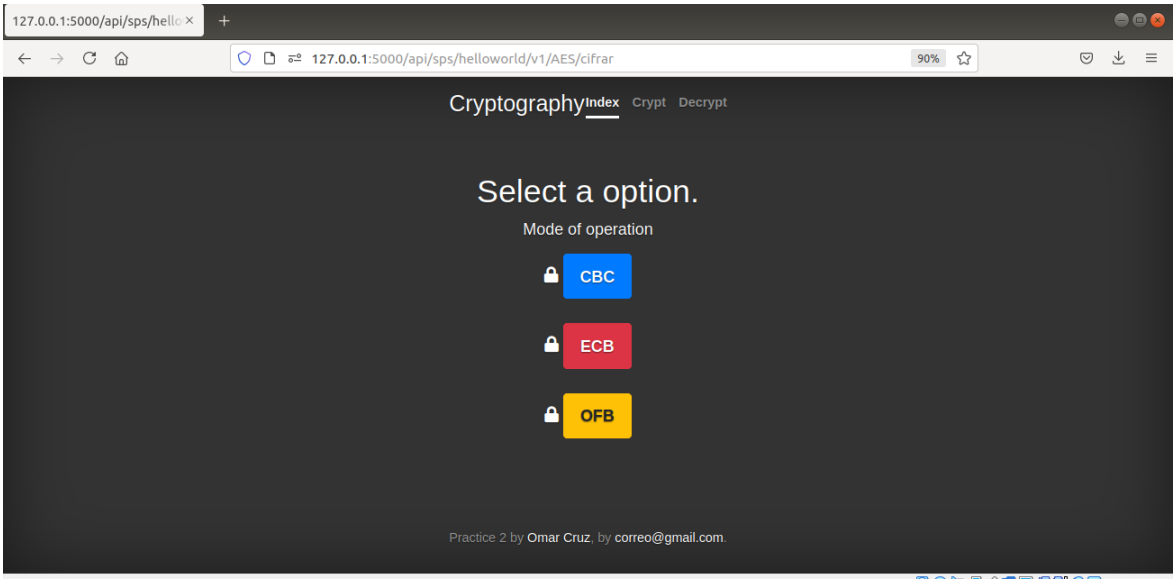

Figura 10: Modos de operación

En la ruta **/AES/{action}/{mode}** con el parametro action=cifrar y mode=CBC, utilizando el método GET

Figura 11: Seleccionar elementos del modo de operación

En la ruta **/AES/{action}/{mode}** con el parametro action=cifrar y mode=CBC, utilizando el método POST, manejamos los resultados del JSON para mostrarlos como se muestra en la figura 12 Ahora solo probamos la ruta **/AES/{action}/{mode}** con el parametro ac-

Figura 12: Muestra de imagen cifrada

tion=descifrar y el parametro mode=CBC, con el método GET, esto con el fin de probar la función de descifrar, en la figura 13 mostramos los valores con o que probaremos ello, en la parte de la imagen seleccionamos la imagen de la salida anterior. la figura 14 muestra la ruta

Figura 13: Seleccionar elementos para el modo de operación

/AES/{action}/{mode} con el parametro action=descifrar y el parametro mode=CBC, utilizando POST

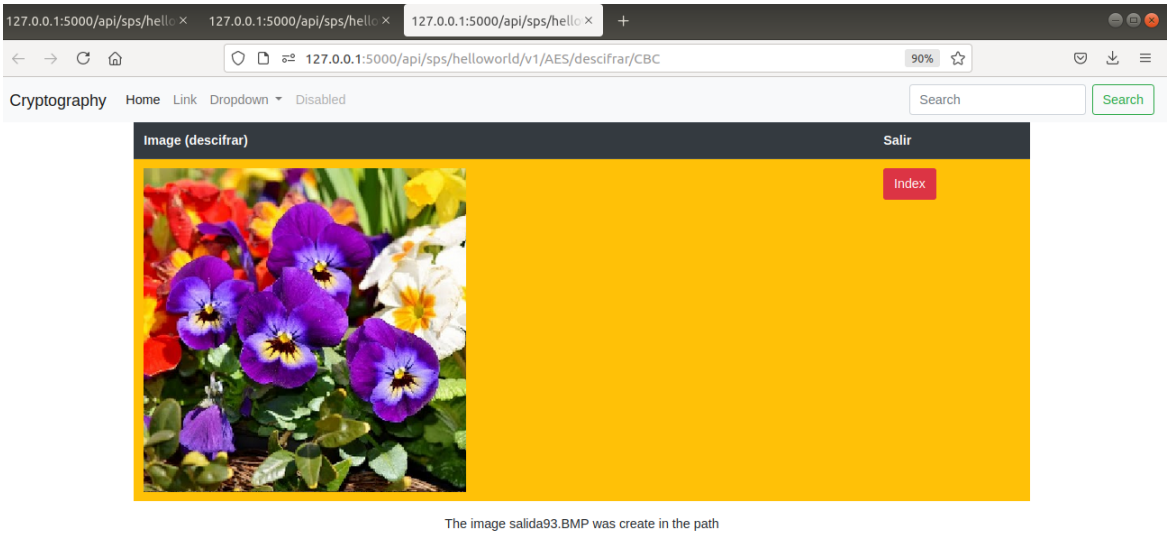

Figura 14: Muestra de imagen descifrada

3 Referencias Bibliográficas

References

[1] ernandez, L., (2015). *Tablas de enrutamiento* Recuperate of:. https://www.redeszone.net/tutoriales/redes-cable/tabla-enrutamiento-router-que-es/

[2] e la Luz, S., (2015). *¿Qué es gns3?* Recuperate of:. https://www.redeszone.net/2015/02/04/gns3-el-conocido-simulador-grafico-de-redes-renueva-su-interfaz-grafica-por-completo/