INFORME PRÁCTICA DE CRIPTOGRAFÍA

ALEXANDRA
STEFANIA
OANE

ÍNDICE

EJERCICIO 1. Tenemos un sistema que usa claves de 16 bytes. Por razones de seguridad vamos a proteger la clave de tal forma que ninguna persona tenga acceso directamente a la clave. Por ello, vamos a realizar un proceso de disociación de la misma, en el cuál tendremos, una clave fija en código, la cual, sólo el desarrollador tendrá acceso, y otra parte en un fichero de propiedades que rellenará el Key Manager. La clave final se generará por código, realizando un XOR entre la que se encuentra en el properties y en el código.

La clave fija en código es A1EF2ABFE1AAEEFF, mientras que en desarrollo sabemos que la clave final (en memoria) es B1AA12BA21AABB12. ¿Qué valor ha puesto el Key Manager en properties para forzar dicha clave final?

Se realiza un XOR con la clave fija y la clave final. El valor que nos devuelve en hexadecimal.

```python
#XOR de los datos binarios
#Creamos la funci'on XOR de dos datos>
def xor(binary_data_1, binary_data_2):
    return bytes([b1^b2 for b1, b2 in zip(binary_data_1, binary_data_2)])
    #Los datos deben pasar a leerse en formato bytes.



if __name__ == "__main__":
    m = bytes("keepcoding", "utf-8")
    k = bytes("1234567891", "utf-8")
    print("El resultado en binario es: ", xor(m,k))
    print("El resultado en hexadecimal es: ", xor(m,k).hex())
```

```
[Running] python -u "/home/kali/Desktop/ejercicospractica/#XOR de los datos binarios.py"
El resultado en binario es:  b'ZWVDVYSQWV'
El resultado en hexadecimal es:  5a575644565953515756

[Done] exited with code=0 in 0.091 seconds
```

```python
#XOR de los datos binarios
#Creamos la funci'on XOR de dos datos>
def xor(binary_data_1, binary_data_2):
    return bytes([b1^b2 for b1, b2 in zip(binary_data_1, binary_data_2)])
    #Los datos deben pasar a leerse en formato bytes.



if __name__ == "__main__":
    m = bytes("A1EF2ABFE1AAEEFF", "utf-8")
    k = bytes("B1AA12BA21AABB12", "utf-8")
    print("El resultado en binario es: ", xor(m,k))
    print("El resultado en hexadecimal es: ", xor(m,k).hex())
```

```
[Running] python -u "/home/kali/Desktop/ejercicospractica/#XOR de los datos binarios.py"
El resultado en binario es:  b'\x03\x00\x04\x07\x03s\x00\x07w\x00\x00\x00\x07\x07wt'
El resultado en hexadecimal es:  03000407037300077700000007077774

[Done] exited with code=0 in 0.043 seconds
```

```python
from operator import xor
from multiplicar import *
def xor_data(binary_data_1, binary_data_2):
    return bytes([b1 ^ b2 for b1, b2 in zip(binary_data_1, binary_data_2)])

m1 = bytes('España',"utf-8")
m = bytes("keepcoding===","utf-8")
k = bytes("abcdefghijddd","utf-8")



print(xor_data(m,k))
print(xor_data(m,k).hex())
print(xor_data(k,m).hex())

#K1 (key manager) ^ K2 (desarrollador) = K (keyStore)



K1= bytes.fromhex('A1EF2ABFE1AAEEFF')
K2= bytes.fromhex('B1AA12BA21AABB12')


K= xor_data(K1,K2)
K_hex = xor_data(K1,K2).hex()

print("K=K1^K2: ", K_hex)

K22 = xor_data(K,K1)

print("K2=K3 ^K1: ", K22.hex())

K1= xor_data(K2,K1)

print("K1=K2 ^ K: ", K1.hex())

```

```
[Running] python -u "/home/vant/Escritorio/ejerciciospractica/multiplicar.py"
b'\n\x07\x06\x14\x06\t\x03\x01\x07\rYYY'
0a07061406090301070d595959
0a07061406090301070d595959
K=K1^K2:  10453805c00055ed
K2=K3 ^K1:  b1aa12ba21aabb12
K1=K2 ^ K:  10453805c00055ed
b'\n\x07\x06\x14\x06\t\x03\x01\x07\rYYY'
0a07061406090301070d595959
0a07061406090301070d595959
K=K1^K2:  10453805c00055ed
K2=K3 ^K1:  b1aa12ba21aabb12
K1=K2 ^ K:  10453805c00055ed

[Done] exited with code=0 in 0.067 seconds
```

home > kali > Desktop > ejercicospractica > #XOR de los datos binarios.py > ...

```python
#XOR de los datos binarios
#Creamos la funcion XOR de dos datos>
def xor(binary_data_1, binary_data_2):
    return bytes([b1^b2 for b1, b2 in zip(binary_data_1, binary_data_2)])
    #Los datos deben pasar a leerse en formato bytes.



if __name__ == "__main__":
    m = bytes("A1EF2ABFE1AAEEFF", "utf-8")
    k = bytes("B1AA12BA21AABB12", "utf-8")
    print("El resultado en binario es: ", xor(m,k))
    print("El resultado en hexadecimal es: ", xor(m,k).hex())
```

```
[Running] python -u "/home/kali/Desktop/ejercicospractica/#XOR de los datos binarios.py"
El resultado en binario es:  b'\x03\x00\x04\x07\x03s\x00\x07w\x00\x00\x00\x07\x07wt'
El resultado en hexadecimal es:  03000407037300077700000007077774

[Done] exited with code=0 in 0.062 seconds
```

La clave fija, recordemos es A1EF2ABFE1AAEEFF, mientras que en producción sabemos que la parte dinámica que se modifica en los ficheros de propiedades es B98A15BA31AEBB22. ¿Qué clave será con la que se trabaje en memoria?

home > kali > Desktop > ejercicospractica > #XOR de los datos binarios.py > ...

```python
#XOR de los datos binarios
#Creamos la funcion XOR de dos datos>
def xor(binary_data_1, binary_data_2):
    return bytes([b1^b2 for b1, b2 in zip(binary_data_1, binary_data_2)])
    #Los datos deben pasar a leerse en formato bytes.



if __name__ == "__main__":
    m = bytes("A1EF2ABFE1AAEEFF", "utf-8")
    k = bytes("B98A15BA31AEBB22", "utf-8")
    print("El resultado en binario es: ", xor(m,k))
    print("El resultado en hexadecimal es: ", xor(m,k).hex())
```

```
[Running] python -u "/home/kali/Desktop/ejercicospractica/#XOR de los datos binarios.py"
El resultado en binario es:  b'\x03\x08}\x07\x03t\x00\x07v\x00\x00\x04\x07\x07tt'
El resultado en hexadecimal es:  03087d07037400077600000407077474

[Done] exited with code=0 in 0.067 seconds
```

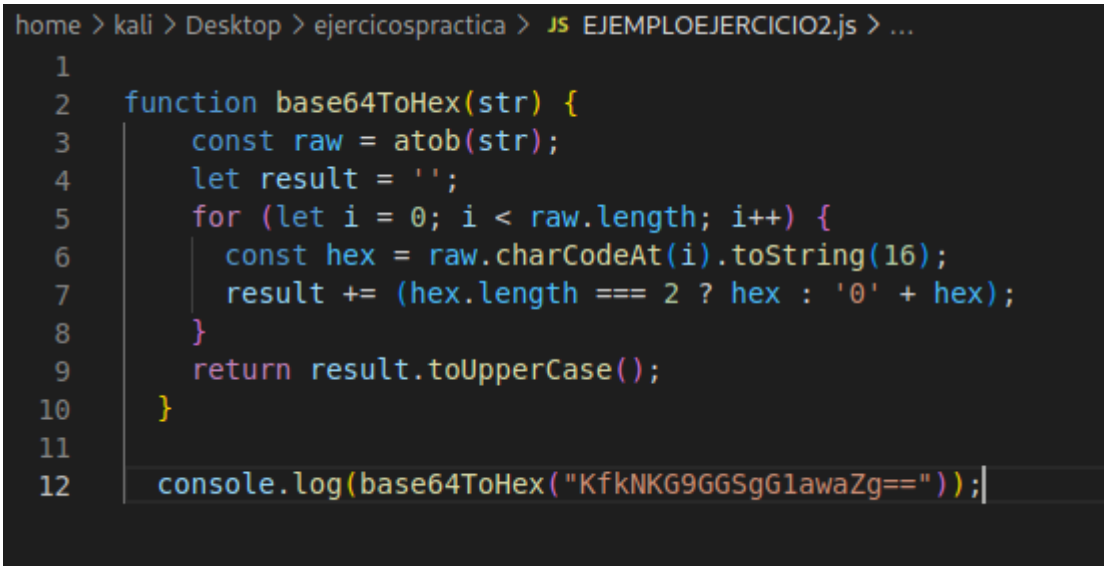
EJERCICIO 2. Dada la clave con etiqueta "cifrado-sim-aes-256" que contiene el keystore. El iv estará compuesto por ceros binarios ("00"). Se requiere obtener el dato en claro correspondiente al siguiente dato cifrado:

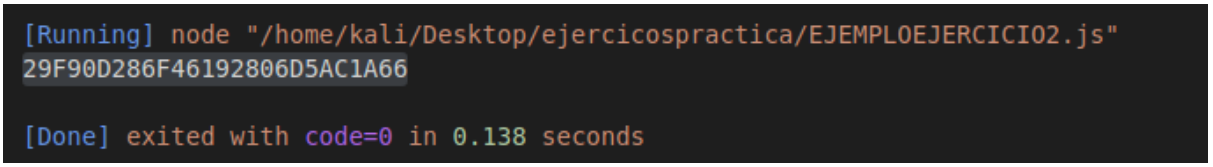
zcFJxR1fzaBj+gVWFRAah1N2wv+G2P01ifrKejlCaGpQkPnZMiexn3WXlGYX5WnNIosy KfkNKG9GGSgG1awaZg==

Para este caso, se ha usado un AES/CBC/PKCS7.

Si lo desciframos, ¿qué obtenemos? ¿Qué ocurre si decidimos cambiar el padding a x923en el descifrado? ¿Cuánto padding se ha añadido en el cifrado? Como truco, estudiar el resultado en hexadecimal, cuando lo imprimas en consola introducir un string para diferenciar el final, como "\|**\|".***

Se valorará positivamente, obtener el dato de la clave desde el keystore mediante codificación en Python (u otro lenguaje).





home > kali > Desktop > criptografia-main > Código fuente > criptografia en bloque > EJERCICIO2.py > ...

```python
from base64 import b64encode, b64decode
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes

#Cliente -> Cifrando el dato, para lo cual, algoritmo (AES/CBC/PKCS7)
#Cifrado
textoPlano_bytes = bytes('', 'UTF-8')
#Se puede generar aleatoriamente una clave de 16 bytes.
#clave = get_random_bytes(16)
clave = bytes.fromhex('979df30474898787a45605ccb9b936d33b780d03cabc81719d52383480dc3120')
iv_bytes = bytes.fromhex('29F90D286F46192806D5AC1A66000000')
cipher = AES.new(clave, AES.MODE_CBC,iv_bytes)
texto_cifrado_bytes = cipher.encrypt(pad(textoPlano_bytes, AES.block_size,  style='pkcs7'))
#Provocamos un fallo por padding incorrecto
#texto_cifrado_bytes = cipher.encrypt(textoPlano_bytes)
#Si se generase de forma automática, por no especificarlo en la llamada, se recuperaría así.
iv_b64 = b64encode(cipher.iv).decode('utf-8')
texto_cifrado_b64 = b64encode(texto_cifrado_bytes).decode('utf-8')
mensaje_json = json.dumps({'iv':iv_b64, 'texto cifrado':texto_cifrado_b64})
print(mensaje_json)

```

```python
#Descifrado
#Servidor, necesita descifrarlo. Recupera el json, y obtiene los campos. Iv entrada, y obtiene el texto cifrado e
try:
    b64 = json.loads(mensaje_json)
    iv_desc_bytes = b64decode(b64['KfkNKG9GGSgG1awaZg=='])
    texto_cifrado_bytes = b64decode(b64['zcFJxR1fzaBj+gVWFRAah1N2wv+G2P01ifrKejICaGpQkPnZMiexn3WXlGYX5WnNIosy'])
    cipher = AES.new(clave, AES.MODE_CBC, iv_desc_bytes)
    mensaje_des_bytes = unpad(cipher.decrypt(texto_cifrado_bytes), AES.block_size, style="pkcs7")
    #mensaje_des_bytes = cipher.decrypt(texto_cifrado_bytes)
    print("En hex: ", mensaje_des_bytes.hex())
    print("El texto en claro es: ", mensaje_des_bytes.decode("utf-8"))

except (ValueError, KeyError) as error:
    print('Problemas para descifrar....')
    print("El motivo del error es: ", error)
```

```
[Running] python -u "/home/kali/Desktop/criptografia-main/Código fuente/criptografia en bloque/EJERCICIO2.py"
{"iv": "KfkNKG9GGSgG1awaZgAAAA==", "texto cifrado": "utpA5dhBD0J3DxddDWDrSw=="}
Problemas para descifrar....
El motivo del error es:  'KfkNKG9GGSgG1awaZg=='

[Done] exited with code=0 in 0.155 seconds
```

home > kali > Desktop > criptografia-main > Código fuente > criptografia en bloque > EJERCICIO2.py > ...

```python

import json
from base64 import b64encode, b64decode
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes

#Cliente -> Cifrando el dato, para lo cual, algoritmo (AES/CBC/PKCS7)
#Cifrado
textoPlano_bytes = bytes('', 'UTF-8')
#Se puede generar aleatoriamente una clave de 16 bytes.
#clave = get_random_bytes(16)
clave = bytes.fromhex('979df30474898787a45605ccb9b936d33b780d03cabc81719d52383480dc3120')
iv_bytes = bytes.fromhex('29F90D286F46192806D5AC1A66000000')
cipher = AES.new(clave, AES.MODE_CBC,iv_bytes)
texto_cifrado_bytes = cipher.encrypt(pad(textoPlano_bytes, AES.block_size,  style='pkcs7'))
#Provocamos un fallo por padding incorrecto
#texto_cifrado_bytes = cipher.encrypt(textoPlano_bytes)
#Si se generase de forma automática, por no especificarlo en la llamada, se recuperaría así.
iv_b64 = b64encode(cipher.iv).decode('utf-8')
texto_cifrado_b64 = b64encode(texto_cifrado_bytes).decode('utf-8')
mensaje_json = json.dumps({'iv':iv_b64, 'texto cifrado':texto_cifrado_b64})
print(mensaje_json)

#Descifrado
#Servidor, necesita descifrarlo. Recupera el json, y obtiene los campos. Iv entrada, y obtiene el
try:
    b64 = json.loads(mensaje_json)
    iv_desc_bytes = b64decode(b64['iv'])
    texto_cifrado_bytes = b64decode(b64['texto cifradd'])
    cipher = AES.new(clave, AES.MODE_CBC, iv_desc_bytes)
    mensaje_des_bytes = unpad(cipher.decrypt(texto_cifrado_bytes), AES.block_size, style="pkcs7")
    #mensaje_des_bytes = cipher.decrypt(texto_cifrado_bytes)
    print("En hex: ", mensaje_des_bytes.hex())
    print("El texto en claro es: ", mensaje_des_bytes.decode("utf-8"))

except (ValueError, KeyError) as error:
    print('Problemas para descifrar....')
    print("El motivo del error es: ", error)
```

```
[Running] python -u "/home/kali/Desktop/criptografia-main/Código fuente/criptografia en bloque/EJERCICIO2.py"
{"iv": "KfkNKG9GGSgG1awaZgAAAA==", "texto cifrado": "utpA5dhBD0J3DxddDWDrSw=="}
En hex:
El texto en claro es:

[Done] exited with code=0 in 0.14 seconds
```

home > kali > Desktop > criptografia-main > Código fuente > criptografia en bloque > EJERCICIO2.py > ...

```python

import json
from base64 import b64encode, b64decode
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes

#Cliente -> Cifrando el dato, para lo cual, algoritmo (AES/CBC/PKCS7)
#Cifrado
textoPlano_bytes = bytes('zcFJxR1fzaBj+gVWFRAah1N2wv+G2P01ifrKejICaGpQkPnZMiexn3WXlGYX5WnNIosy', 'UTF-8')
#Se puede generar aleatoriamente una clave de 16 bytes.
#clave = get_random_bytes(16)
clave = bytes.fromhex('e2cff885901a5449e9c448ba5b948a8c4ee377152b3f1acfa0148fb3a426db72')
iv_bytes = bytes.fromhex('29F90D286F46192806D5AC1A66000000')
cipher = AES.new(clave, AES.MODE_CBC,iv_bytes)
texto_cifrado_bytes = cipher.encrypt(pad(textoPlano_bytes, AES.block_size,  style='pkcs7'))
#Provocamos un fallo por padding incorrecto
#texto_cifrado_bytes = cipher.encrypt(textoPlano_bytes)
#Si se generase de forma automática, por no especificarlo en la llamada, se recuperaría así.
iv_b64 = b64encode(cipher.iv).decode('utf-8')
texto_cifrado_b64 = b64encode(texto_cifrado_bytes).decode('utf-8')
mensaje_json = json.dumps({'iv':iv_b64, 'texto cifrado':texto_cifrado_b64})
print(mensaje_json)

#Descifrado
#Servidor, necesita descifrarlo. Recupera el json, y obtiene los campos. Iv entrada, y obtiene el texto cifra
try:
    b64 = json.loads(mensaje_json)
    iv_desc_bytes = b64decode(b64['iv'])
    texto_cifrado_bytes = b64decode(b64['texto cifrado'])
    cipher = AES.new(clave, AES.MODE_CBC, iv_desc_bytes)
    mensaje_des_bytes = unpad(cipher.decrypt(texto_cifrado_bytes), AES.block_size, style="pkcs7")
    #mensaje_des_bytes = cipher.decrypt(texto_cifrado_bytes)
    print("En hex: ", mensaje_des_bytes.hex())
    print("El texto en claro es: ", mensaje_des_bytes.decode("utf-8"))

except (ValueError, KeyError) as error:
    print('Problemas para descifrar....')
    print("El motivo del error es: ", error)
```

```
[Running] python -u "/home/kali/Desktop/criptografia-main/Código fuente/criptografia en bloque/EJERCICIO2.py"
{"iv": "KfkNKG9GGSgG1awaZgAAAA==", "texto cifrado": "bYsqR1H20te+m6QCMVzj//G1NE3/2R/mP/3y0FZLr6eOK3MuvqXyDcpvCxAA9wjl6c9O36v86C4P9j/X+41y2aDWOgYuiXkMlVzSIjw6jsU="}
En hex:  7a63464a785231667a61426a2b6756574652416168314e3277762b47325030316966724b656a4943614770516b506e5a4d6965786e3357586c47595835576e4e496f7379
El texto en claro es:  zcFJxR1fzaBj+gVWFRAah1N2wv+G2P01ifrKejICaGpQkPnZMiexn3WXlGYX5WnNIosy

[Done] exited with code=0 in 0.163 seconds
```

```python

import json
from base64 import b64encode, b64decode
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes

#Cliente -> Cifrando el dato, para lo cual, algoritmo (AES/CBC/PKCS7)
#Cifrado
textoCifrado_bytes = bytes('zcFJxR1fzaBj+gVWFRAah1N2wv+G2P01ifrKejICaGpQkPnZMiexn3WXlGYX5WnNIosy', 'UTF-8')
#Se puede generar aleatoriamente una clave de 16 bytes.
#clave = get_random_bytes(16)
clave = bytes.fromhex('e2cff885901a5449e9c448ba5b948a8c4ee377152b3f1acfa0148fb3a426db72')
iv_bytes = bytes.fromhex('29F90D286F46192806D5AC1A66000000')
cipher = AES.new(clave, AES.MODE_CBC,iv_bytes)
texto_descifrado_bytes = cipher.decrypt(pad(textoCifrado_bytes, AES.block_size,  style='pkcs7'))
#Provocamos un fallo por padding incorrecto
#texto_cifrado_bytes = cipher.encrypt(textoPlano_bytes)
#Si se generase de forma automática, por no especificarlo en la llamada, se recuperaría así.
iv_b64 = b64encode(cipher.iv).decode('utf-8')
texto_cifrado_b64 = b64encode(texto_descifrado_bytes).decode('utf-8')
mensaje_json = json.dumps({'iv':iv_b64, 'texto cifrado':texto_cifrado_b64})
print(mensaje_json)

#Descifrado
#Servidor, necesita descifrarlo. Recupera el json, y obtiene los campos. Iv entrada, y obtiene el texto cif
try:
    b64 = json.loads(mensaje_json)
    iv_desc_bytes = b64decode(b64['iv'])
    texto_descifrado_bytes = b64decode(b64['texto cifrado'])
    cipher = AES.new(clave, AES.MODE_CBC, iv_desc_bytes)
    mensaje_des_bytes = unpad(cipher.decrypt(texto_descifrado_bytes), AES.block_size, style="pkcs7")
    #mensaje_des_bytes = cipher.decrypt(texto_cifrado_bytes)
    print("En hex: ", mensaje_des_bytes.hex())
    print("El texto en claro es: ", mensaje_des_bytes.decode("utf-8"))
```

```python
#Descifrado
#Servidor, necesita descifrarlo. Recupera el json, y obtiene los campos. Iv entrada, y obtiene el texto cifrado entrada. Aplica el algoritmo (AES/CBC/PKCS7)
try:
    b64 = json.loads(mensaje_json)
    iv_desc_bytes = b64decode(b64['iv'])
    texto_descifrado_bytes = b64decode(b64['texto cifrado'])
    cipher = AES.new(clave, AES.MODE_CBC, iv_desc_bytes)
    mensaje_des_bytes = unpad(cipher.decrypt(texto_descifrado_bytes), AES.block_size, style="pkcs7")
    #mensaje_des_bytes = cipher.decrypt(texto_cifrado_bytes)
    print("En hex: ", mensaje_des_bytes.hex())
    print("El texto en claro es: ", mensaje_des_bytes.decode("utf-8"))

except (ValueError, KeyError) as error:
    print('Problemas para descifrar....')
    print("El motivo del error es: ", error)
```

```
[Running] python -u "/home/kali/Desktop/criptografia-main/Código fuente/criptografia en bloque/EJERCICIO2.py"
{"iv": "KfkNKG9GGSgG1awaZgAAAA==", "texto cifrado": "tQuWDmsB31Cmcr7+jYtqP1WFS7pVlTBJ2lX04X7Jg//4M0/zrsU7zRfME3VYifbXui/F2lGvjbTGVIsJybgEcYtqR+gkgxLg/s5X3256UQA="}
Problemas para descifrar....
El motivo del error es:  Padding is incorrect.

[Done] exited with code=0 in 0.256 seconds
```

EJERCICIO 3. Se requiere cifrar el texto “*Este curso es de lo mejor que podemos encontrar en el mercado*”. La clave para ello, tiene la etiqueta en el keyStore “cifrado-sim-chacha-256”. El nonce “*9Yccn/f5nJJhAt2S*”. 979df30474898787a45605ccb9b36d33b780d03cabc81719d52383480dc3120

¿Cómo podríamos mejorar de forma sencilla el sistema, de tal forma, que no sólo garanticemos la confidencialidad sino, además, la integridad del mismo? Demuestra, tu propuesta por código, así como añade los datos necesarios para evaluar tu solución.

home > kali > Desktop > ejercicospractica > EJERCICO3.py > ...

```python
import json
from base64 import b64encode, b64decode
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes

#Cifrado
textoPlano_bytes = bytes('Este curso es de lo mejor que podemos encontrar en el mercado', 'UTF-8')

#clave = get_random_bytes(16)
clave = bytes.fromhex('979df30474898787a45605ccb9b936d33b780d03cabc81719d52383480dc3120')
nonce = bytes.fromhex('f5871c9ff7f99c926102dd92')
cipher = AES.new(clave, AES.MODE_CTR,nonce=nonce)
texto_cifrado_bytes = cipher.encrypt(textoPlano_bytes)

nonce_b64 = b64encode(cipher.nonce).decode('utf-8')
texto_cifrado_b64 = b64encode(texto_cifrado_bytes).decode('utf-8')
mensaje_json = json.dumps({'nonce':nonce_b64, 'texto cifrado':texto_cifrado_b64})
print(mensaje_json)

#Descifrado

try:
    b64 = json.loads(mensaje_json)
    nonce_desc_bytes = b64decode(b64['nonce'])
    texto_cifrado_bytes = b64decode(b64['texto cifrado'])
    cipher = AES.new(clave, AES.MODE_CTR, nonce=nonce_desc_bytes)
    mensaje_des_bytes = cipher.decrypt(texto_cifrado_bytes)
    print("El texto en claro es: ", mensaje_des_bytes.decode("utf-8"))

except (ValueError, KeyError) as error:
    print('Problemas para descifrar....')
    print("El motivo del error es: ", error)
```

```
[Running] python -u "/home/kali/Desktop/ejercicospractica/EJERCICO3.py"
{"nonce": "9Yccn/f5nJJhAt2S", "texto cifrado": "nNEpHk9g0Mh55iENpj2SJyLkdTDa/lEgKehRG268VwCTG3FYxxQ92aOdwQPouppMcEHafItHyHsnCybDkA=="}
El texto en claro es:  Este curso es de lo mejor que podemos encontrar en el mercado

[Done] exited with code=0 in 0.13 seconds
```

EJERCICIO 4. Tenemos el siguiente jwt, cuya clave es “KeepCoding”.

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmlvIjoiRmVsaXBlIFJvZHJcdTAwZWRndWV6Iiwicm9sIjoiaXNOb3JtYWwifQ.-KiAA8cjkamjwRUiNVHgGeJU8k2wiErdxQP_iFXumM8

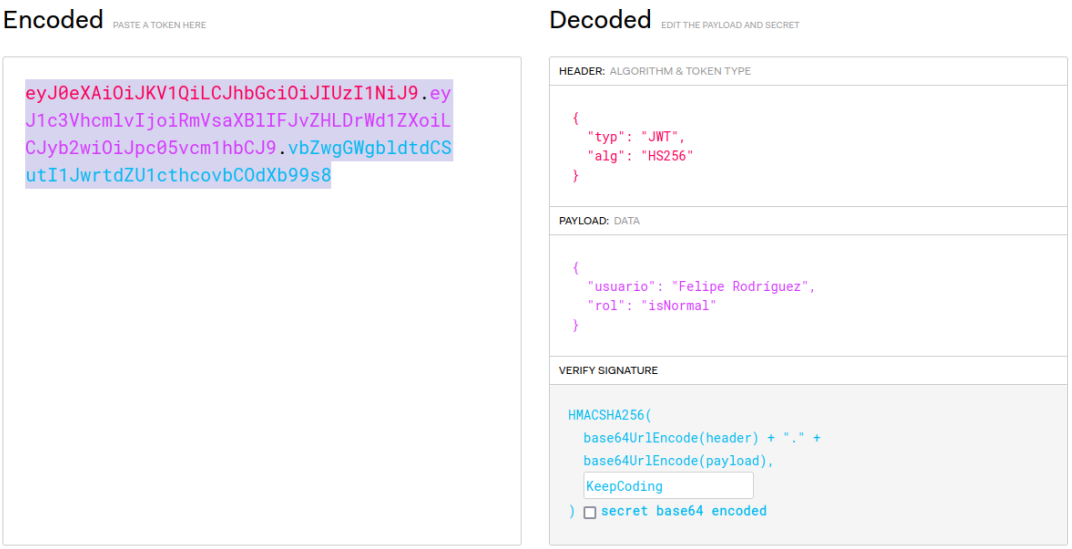
¿Qué algoritmo de firma hemos realizado?

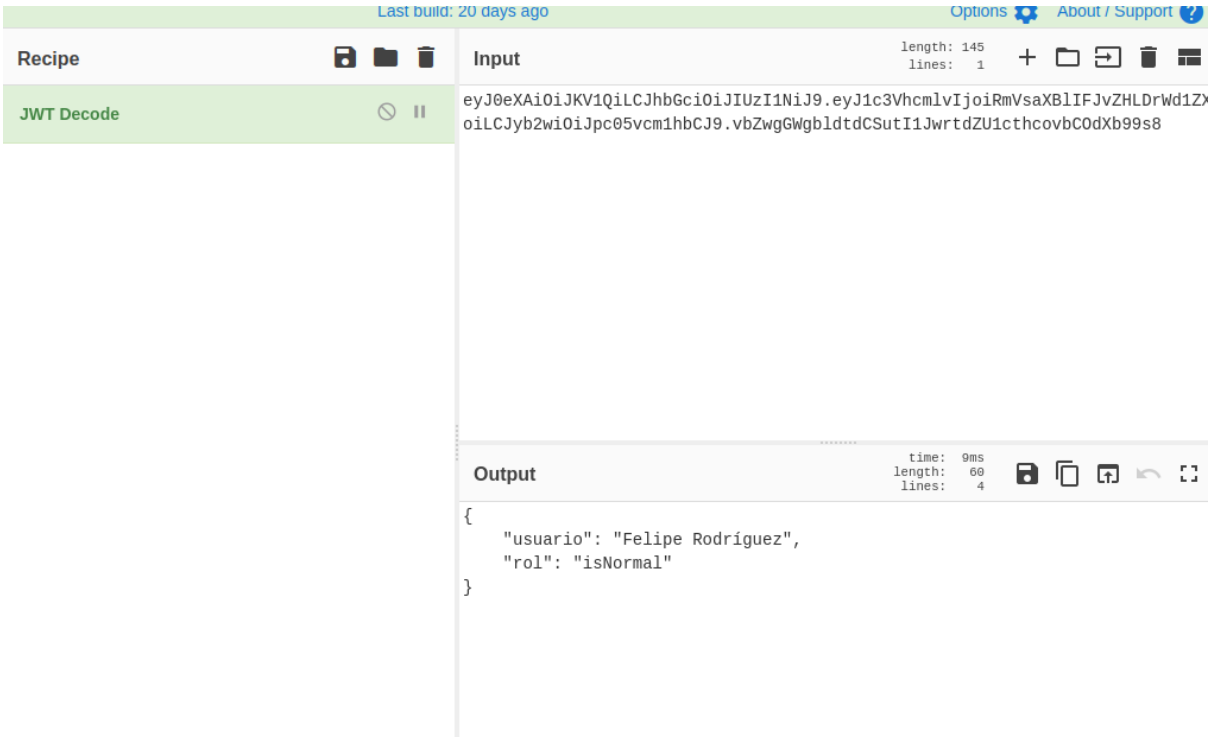
El algoritmo realizado en este ejercicio es el SHA2-256, tipo JWT.

¿Cuál es el body del jwt?

En el body del jwt se muestra los siguientes datos:
usuario: Felipe Rodríguez
rol: isNormal

Se han usado las herramientas de Cyberchef y Jwt.io.





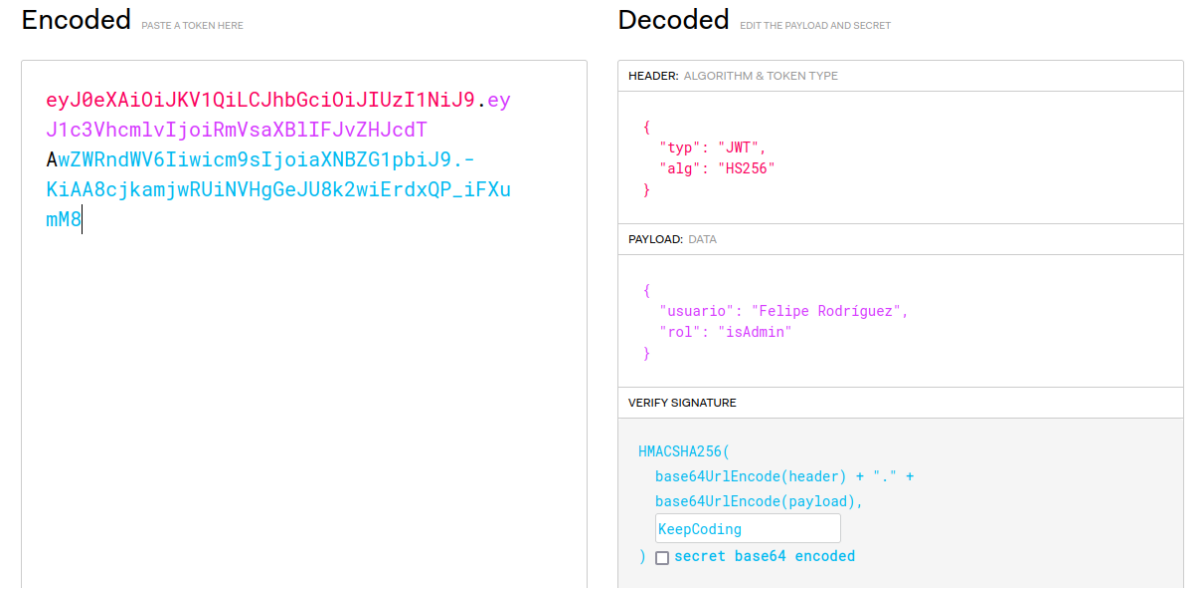
Un hacker está enviando a nuestro sistema el siguiente jwt:

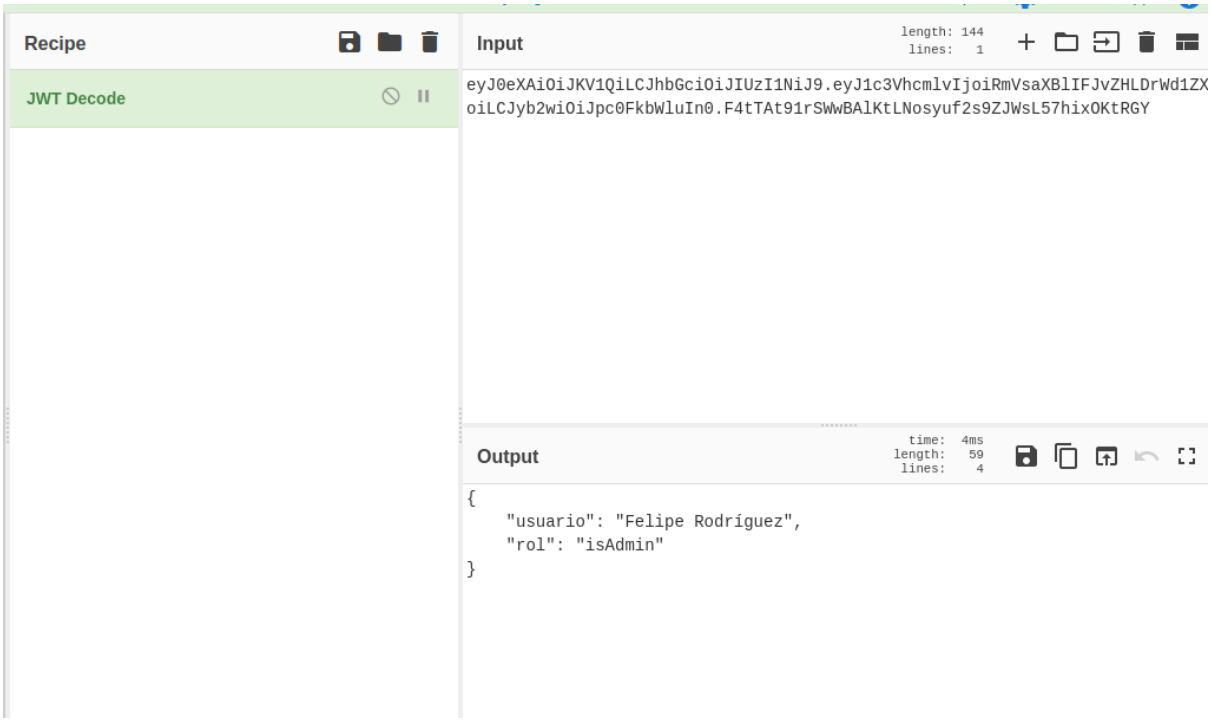
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmlvIjoiRmVsaXBlIFJvZHJcdTAwZWRndWV6Iiwicm9sIjoiaXNBZG1pbiJ9.-KiAA8cjkamjwRUiNVHgGeJU8k2wiErdxQP_iFXumM8

¿Qué está intentando realizar?

Está intentando cambiar el rol de usuario normal a rol de administrador.

¿Qué ocurre si intentamos validarlo con pyjwt?





Si se intenta validar, nos dará error debido al cambio realizado.

EJERCICIO 5. El siguiente hash se corresponde con un SHA3 Keccak del texto "En KeepCoding aprendemos cómo protegernos con criptografía".

bced1be95fbd85d2ffcce9c85434d79aa26f24ce82fbd4439517ea3f072d56fe

¿Qué tipo de SHA3 hemos generado?

Se ha generado un SHA3-256.

home > kali > Desktop > ejercicospractica > EJERCICIO5.py > ...

```python
import hashlib


# initiating the "s" object to use the
# sha3_256 algorithm from the hashlib module.
s = hashlib.sha3_256()

# will output the name of the hashing algorithm currently in use.
print(s.name)

# will output the Digest-Size of the hashing algorithm being used.
print(s.digest_size)

# providing the input to the hashing algorithm.
s.update(bytes("En KeepCoding aprendemos cómo protegernos con criptografía","UTF-8"))

print(s.hexdigest())
```

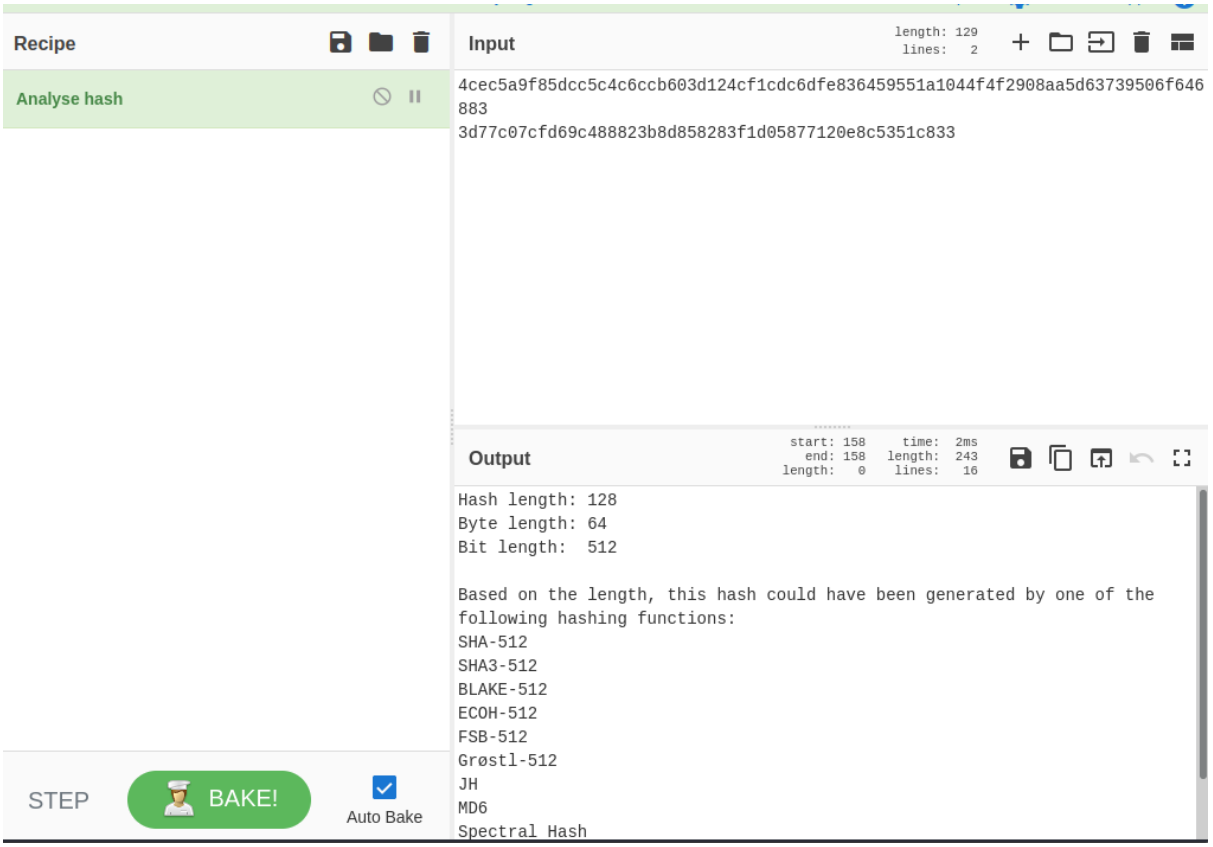
```
[Running] python -u "/home/kali/Desktop/ejercicospractica/EJERCICIO5.py"
sha3_256
32
bced1be95fbd85d2ffcce9c85434d79aa26f24ce82fbd4439517ea3f072d56fe

[Done] exited with code=0 in 0.051 seconds
```

Y si hacemos un SHA2, y obtenemos el siguiente resultado:

4cec5a9f85dcc5c4c6ccb603d124cf1cdc6dfe836459551a1044f4f2908aa5d63739506f646883 3d77c07cfd69c488823b8d858283f1d05877120e8c5351c833

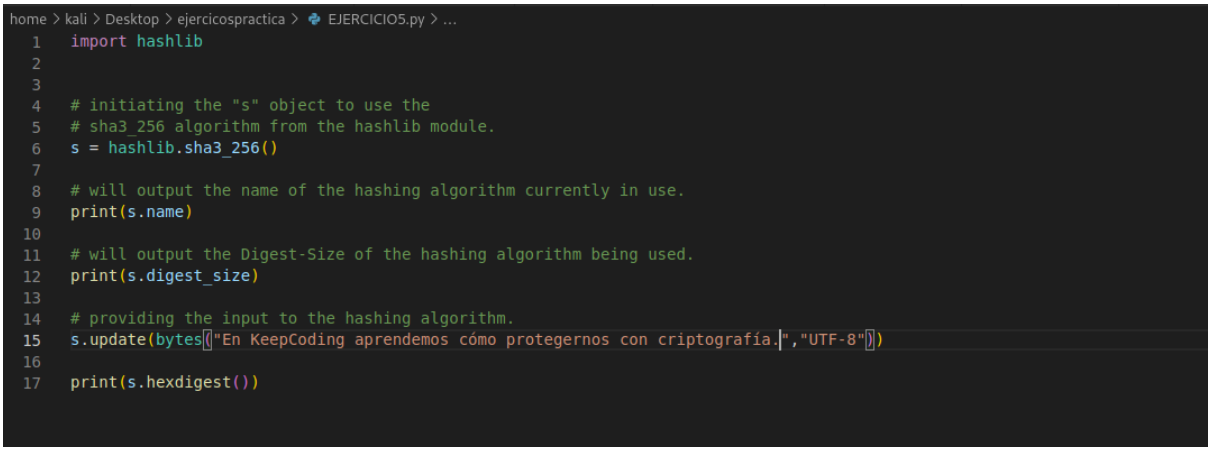
¿Qué hash hemos realizado?

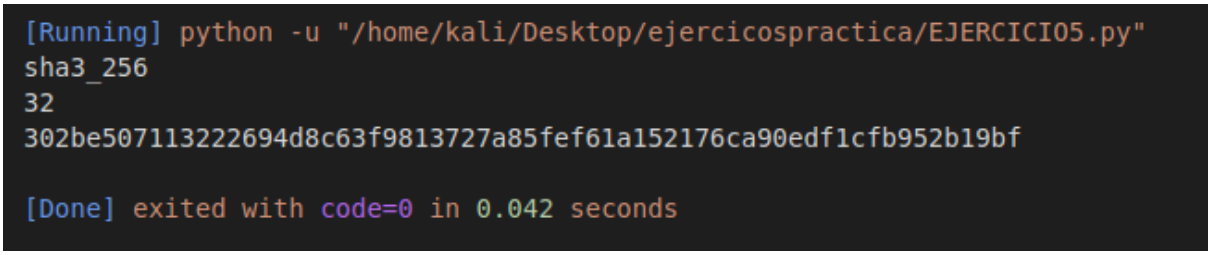
Con la herramienta Cyberchef se ha realizado un análisis del tipo de hash.



Se observa que el tamaño del hash es de 128 bits y es SHA3-512.

Genera ahora un SHA3 Keccak de 256 bits con el siguiente texto: "En KeepCoding aprendemos cómo protegernos con criptografía." ¿Qué propiedad destacarías del hash, atendiendo a los resultados anteriores?





En comparación con el texto y resultado anterior, la propiedad a destacar es el signo de puntuación al final del texto ".". Esto provoca que el nuevo resultado generado con un SHA3 Keccak de 256 bits sea distinto al anterior.

Cada vez que se añade un caracter nuevo en el texto inicial, automáticamente el hash cambia.

EJERCICIO 6. Calcula el hmac-256 (usando la clave contenida en el KeyStore) del siguiente texto:

Siempre existe más de una forma de hacerlo, y más de una solución válida.

Se debe evidenciar la respuesta. Cuidado si se usan herramientas fuera de los lenguajes de programación, por las codificaciones es mejor trabajar en hexadecimal.

home > kali > Desktop > ejercicospractica > JS EJERCICIO6.js > ...

```js
const crypto = require("crypto");

var algoritmo = "sha256";
var mensaje = "Siempre existe más de una forma de hacerlo, y más de una solución válida.";
var key = "2712a51c997e14b4df08d55967641b0677ca31e049e672a4b06861aa4d5826eb";

var hash = crypto.createHmac(algoritmo, key);

hash.update(mensaje);
console.log("Mensaje:\t", mensaje);
console.log("Clave:\t\t", key);
console.log("Método:\t\t", algoritmo);
var miHash = hash.digest("hex");
console.log("\nHMAC es:\t", miHash);
hash = crypto.createHash(algoritmo);
console.log("HMAC es:\t", hash.digest("base64"));
console.log("Longitud of HMAC: ", miHash.length * 4, " bits");
```

```
[Running] node "/home/kali/Desktop/criptografia-main/Código fuente/Hashing y Authentication/HMAC256.js"
Mensaje:     Siempre existe más de una forma de hacerlo, y más de una solución válida.
Clave:       2712a51c997e14b4df08d55967641b0677ca31e049e672a4b06861aa4d5826eb
Método:      sha256

HMAC es:     0358b0a81f64bfbe9dcf57e9a0f5f155f41cd5bacaef477e63aa23e2f29082ee
HMAC es:     47DEQpj8HBSa+/TImW+5JCeuQeRkm5NMpJWZG3hSuFU=
Longitud of HMAC:  256  bits
```

home > kali > Desktop > ejercicospractica > ejercico6.1.py > ...

```python
from Crypto.Hash import HMAC, SHA256

secret = bytes('2712a51c997e14b4df08d55967641b0677ca31e049e672a4b06861aa4d5826eb', 'utf-8')

#Generación
msg0= bytes('Siempre existe más de una forma de hacerlo, y más de una solución válida.', 'utf-8')
h = HMAC.new(secret, msg=msg0, digestmod=SHA256)
#h.update(bytes('Siempre existe más de una forma de hacerlo, y más de una solución válida.', 'utf-8'))
print(h.hexdigest())
mac = h.hexdigest()

#Verificación
h2 = HMAC.new(secret, digestmod=SHA256)
msg = bytes('Siempre existe más de una forma de hacerlo, y más de una solución válida.', 'utf-8')
h2.update(msg)
try:
    h2.hexverify(mac)
    print("Mensaje validado ok")
except ValueError:
    print("Mensaje validado ko")
```

```
[Running] python -u "/home/kali/Desktop/ejercicospractica/ejercico6.1.py"
0358b0a81f64bfbe9dcf57e9a0f5f155f41cd5bacaef477e63aa23e2f29082ee
Mensaje validado ok

[Done] exited with code=0 in 0.1 seconds
```

EJERCICIO 7. Trabajamos en una empresa de desarrollo que tiene una aplicación web, la cual requiere un login y trabajar con passwords. Nos preguntan qué mecanismo de almacenamiento de las mismas proponemos.
Tras realizar un análisis, el analista de seguridad propone un hash SHA-1. Su responsable, le indica que es una mala opción. ¿Por qué crees que es una mala opción?
El SHA-1 es una mala opción para utilizarlo como mecanismo de almacenamiento debido a que se considera débil. Produce un valor de hash de 160 bits.
Se descubrió que este tipo de hash tiene varios fallos teóricos que podría provocar ataques de colisión. Es una situación muy complicada ya que permitiría que los atacantes cambiar archivos, sin que se observe.

Después de meditarlo, propone almacenarlo con un SHA-256, y su responsable le pregunta si no lo va a fortalecer de alguna forma. ¿Qué se te ocurre?

Como se comunica en el enunciado que se propone el almacenamiento de las contraseñas con SHA-256, una de las formas para fortalecer el hash, sería:
SHA-256 + salt.
Un salt es un valor generado por una función criptográficamente segura que se agrega a la entrada de funciones hash para crear hashes únicos para cada entrada, independientemente de que la entrada no sea única. Un salt hace que una función hash parezca no determinista, lo cual es bueno ya que no queremos revelar contraseñas duplicadas a través de nuestro hash.
Es importante utilizar sales únicas, por ejemplo si varios usuarios utilizaran la misma contraseña, ambas contraseñas que tuviesen salt tendrían el “mismo valor”, pero si se elige otro salt para la misma contraseña, se obtienen varias contraseñas únicas y más largas que tienen un valor diferente.

El almacenamiento de contraseñas sería:

Salt+ Hash + Nombre de usuario

Cuando el usuario inicie sesión se puede buscar por el nombre de usuario, agregar el salt a la contraseña proporcionada, codificarlo y verificar si el hash almacenado coincide con el hash calculado.

Es muy importante que el salt sea único para cada hash. Además si el almacenamiento lo permite usar Salt de 32 y 64 bytes con el tamaño real dependiendo de la función de protección que tenga. Cuanto más larga sea el salt aumenta la complejidad computacional de contraseñas de posible ataque.

Las contraseñas se codifican y se añade el salt con el algoritmo “bcrypt”.

Parece que el responsable se ha quedado conforme, tras mejorar la propuesta del SHA256, no obstante, hay margen de mejora. ¿Qué propondrías?

Actualmente es un hash que no ha sido roto y ofrece seguridad, además de ser más rápido que otros hashes. Pero se podría mejorar con otro tipo de hash.

Un inconveniente que tiene SHA-2 es que no puede generar MACs, sin embargo SHA-3 no tiene esta dificultad.

EJERCICIO 8. Tenemos la siguiente API REST, muy simple.

Request:

Post /movimientos

Campo	Tipo	Requiere Confidencialidad	Observaciones
idUsuario	Number	N	Identificador
Usuario	String	S	Nombre y Apellidos
Tarjeta	Number	S	

Petición de ejemplo que se desea enviar:

{"idUsuario":1,"usuario":"José Manuel Barrio Barrio","tarjeta":4231212345676891}

Response:

Campo	Tipo	Requiere Confidencialidad	Observaciones
idUsuario	Number	N	Identificador
movTarjeta	Array	S	Formato del ejemplo
Saldo	Number	S	Tendra formato 12300 para indicar 123.00
Moneda	String	N	EUR, DOLLAR

{
"idUsuario": 1,
"movTarjeta": [{
"id": 1,
"comercio": "Comercio Juan",
"importe": 5000
}, {
"id": 2,
"comercio": "Rest Paquito",
"importe": 6000
}],
"Moneda": "EUR",
"Saldo": 23400
}

Como se puede ver en el API, tenemos ciertos parámetros que deben mantenerse confidenciales. Así mismo, nos gustaría que nadie nos modificase el mensaje sin que nos enterásemos. Se requiere una redefinición de dicha API para garantizar la integridad y la confidencialidad de los mensajes. Se debe asumir que el sistema end to end no usa TLS entre todos los puntos.

¿Qué algoritmos usarías? ¿Serías capaz de hacer un ejemplo en Python de cómo resolverlo?
Se usaría un algoritmo HS256. Se ha intentado pero no ha sido posible.

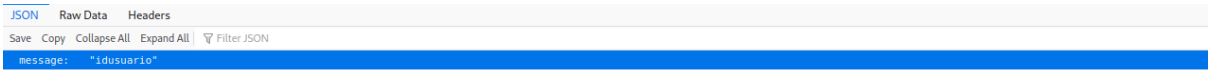
Importante, de cara a calcular cualquier cosa, se deben quitar retornos de carro, espacios intermedios, usar las mismas comillas, etc)

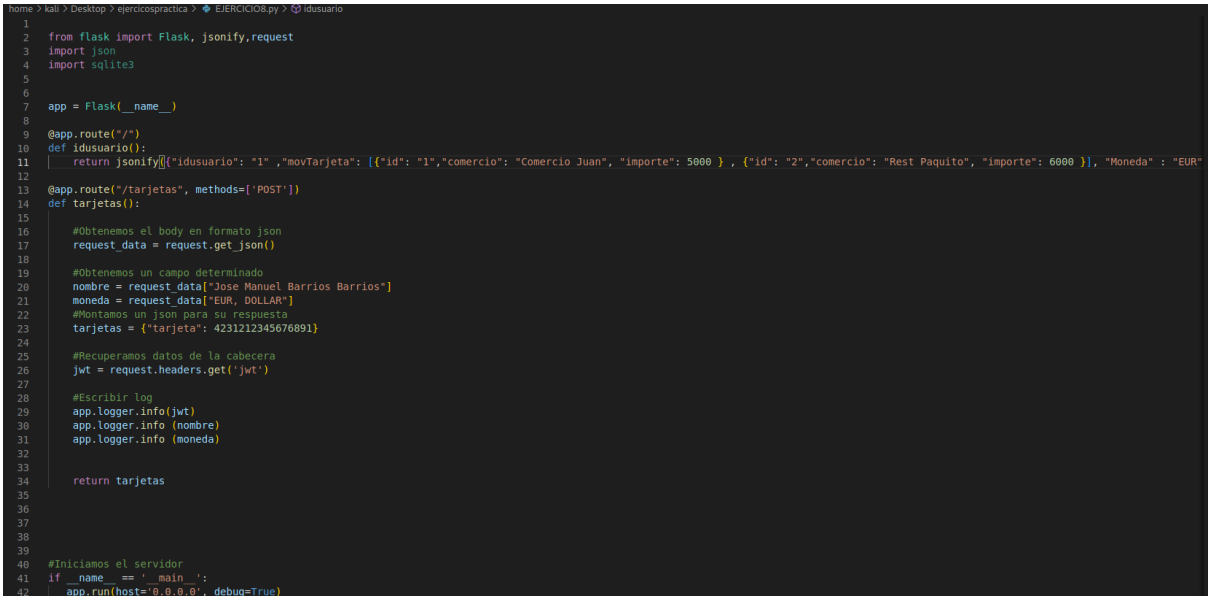
En el siguiente apartado para crear una API sencilla con los parámetros confidenciales, con la ayuda del ejemplo del archivo “servidor.py”.
Se abrirá un entorno virtual

home > kali > Desktop > ejercicospractica > EJERCICIO8.py > idusuario

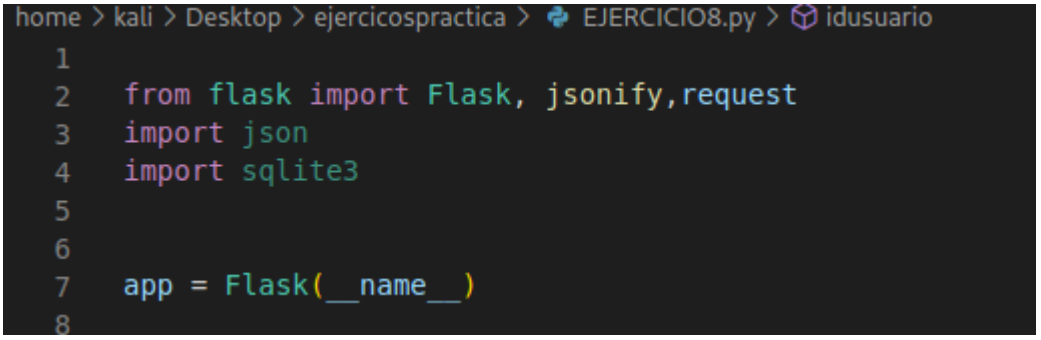
```python

from flask import Flask, jsonify,request
import json
import sqlite3


app = Flask(__name__)

@app.route("/")
def idusuario():
    return jsonify({"message": "idusuario"})

@app.route("/tarjetas", methods=['POST'])
def tarjetas():

    #Obtenemos el body en formato json
    request_data = request.get_json()

    #Obtenemos un campo determinado
    nombre = request_data["Jose Manuel Barrios Barrios"]

    #Montamos un json para su respuesta
    tarjetas = {"tarjeta": 4231212345676891}

    #Recuperamos datos de la cabecera
    jwt = request.headers.get('jwt')

    #Escribir log
    app.logger.info(jwt)
    app.logger.info (nombre)


    return tarjetas




#Iniciamos el servidor
if __name__ == '__main__':
   app.run(host='0.0.0.0', debug=True)
```

```
[Running] python -u "/home/kali/Desktop/ejercicospractica/EJERCICIO8.py"
 * Serving Flask app 'EJERCICIO8' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://10.0.2.15:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 111-639-658
10.0.2.15 - - [31/Jul/2022 10:34:06] "GET / HTTP/1.1" 200 -
```





En la captura completa no se puede ver correctamente, a continuación se muestran las capturas por partes:

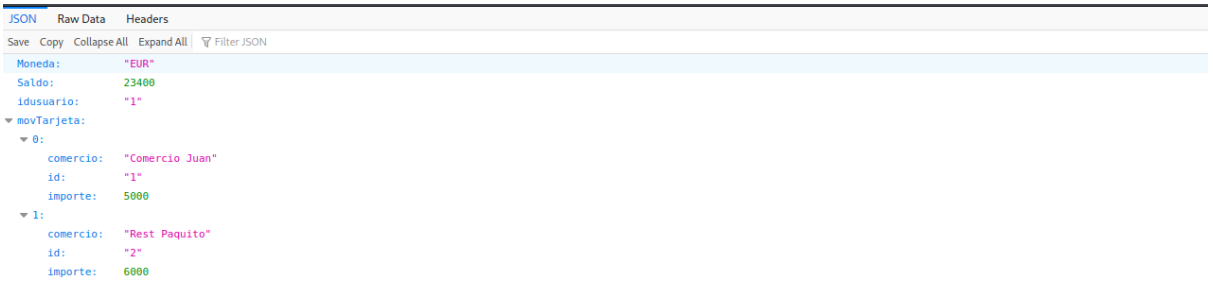


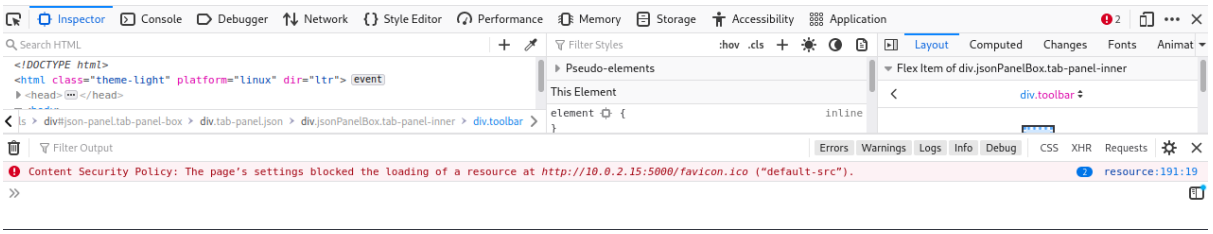
```
6
7   app = Flask(__name__)
8
9   @app.route("/")
0   def idusuario():
1       return jsonify({"idusuario": "1" ,"movTarjeta": [{"id": "1","comercio": "Comercio Juan", "importe": 5000 }
2
```

```
, {"id": "2","comercio": "Rest Paquito", "importe": 6000 }], "Moneda" : "EUR","Saldo" : 23400})
```

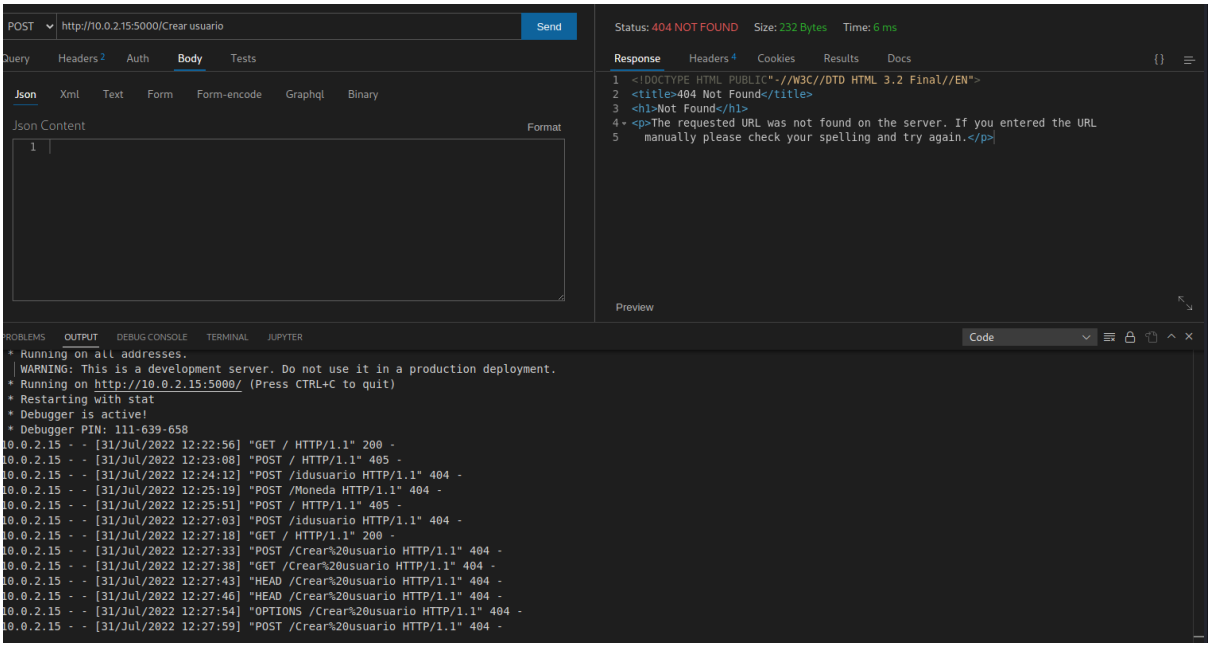
```
13  @app.route("/tarjetas", methods=['POST'])
14  def tarjetas():
15
16      #Obtenemos el body en formato json
17      request_data = request.get_json()
18
19      #Obtenemos un campo determinado
20      nombre = request_data["Jose Manuel Barrios Barr
21      moneda = request_data["EUR, DOLLAR"]
22      #Montamos un json para su respuesta
23      tarjetas = {"tarjeta": 4231212345676891}
24
25      #Recuperamos datos de la cabecera
26      jwt = request.headers.get('jwt')
27
28      #Escribir log
29      app.logger.info(jwt)
30      app.logger.info (nombre)
31      app.logger.info (moneda)
32
33
34      return tarjetas
35
36
37
38
39
40  #Iniciamos el servidor
41  if __name__ == '__main__':
42      app.run(host='0.0.0.0', debug=True)
```

```
[Running] python -u "/home/kali/Desktop/ejercicospractica/EJERCICIO8.py"
 * Serving Flask app 'EJERCICIO8' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://10.0.2.15:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 111-639-658
10.0.2.15 - - [31/Jul/2022 11:08:02] "GET / HTTP/1.1" 200 -
```





JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON
Moneda: "EUR"
Saldo: 23400
idusuario: "1"
movTarjeta:
0:
comercio: "Comercio Juan"
id: "1"
importe: 5000
1:
comercio: "Rest Paquito"
id: "2"
importe: 6000



EJERCICIO 9. El responsable de Raúl, Pedro, ha enviado este mensaje a RRHH:

Se debe ascender inmediatamente a Raúl. Es necesario mejorarle sus condiciones económicas un 20% para que se quede con nosotros.

Lo acompaña del siguiente fichero de firma PGP (MensajeRespoDeRaulARRHH.txt.sig). Nosotros, que pertenecemos a RRHH vamos al directorio a recuperar la clave para verificarlo. Tendremos los ficheros Pedro-priv.txt y Pedro-publ.txt, con las claves privada y pública.

Las claves de los ficheros de RRHH son RRHH-priv.txt y RRHH-publ.txt que también se tendrán disponibles.

Se requiere verificar la misma, y evidenciar dicha prueba.

Así mismo, se requiere firmar el siguiente mensaje con la clave correspondiente de las anteriores, simulando que eres personal de RRHH.

Viendo su perfil en el mercado, hemos decidido ascenderle y mejorarle un 25% su salario. Saludos.

Por último, cifra el siguiente mensaje tanto con la clave pública de RRHH como la de Pedro y adjunta el fichero con la práctica.

Estamos todos de acuerdo, el ascenso será el mes que viene.

EJERCICIO 10. Nuestra compañía tiene un contrato con una empresa que nos da un servicio de almacenamiento de información de videollamadas. Para lo cual, la misma nos envía la clave simétrica de cada videollamada cifrada usando un RSA-OAEP. El hash que usa el algoritmo interno es un SHA-256.

El texto cifrado es el siguiente:

7edee3ec0b808c440078d63ee65b17e85f0c1adbc0da1b7fa842f24fb06b332c1560 38062d9daa8ccfe83bace1dca475cfb7757f1f6446840044fe698a631fe882e1a6fc 00a2de30025e9dcc76e74f9d9d721e9664a6319eaa59dc9011bfc624d2a63eb0e449 ed4471ff06c9a303465d0a50ae0a8e5418a1d12e9392faaaf9d4046aa16e424ae1e2 6844bcf4abc4f8413961396f2ef9ffcd432928d428c2a23fb85b497d89190e3cfa49 6b6016cd32e816336cad7784989af89ff853a3acd796813eade65ca3a10bbf58c621 5fdf26ce061d19b39670481d03b51bb0eecc926c9d6e9cb05ba56082a899f9aa72f9 4c158e56335c5594fcc7f8f301ac1e15a938

Las claves pública y privada las tenemos en los ficheros clave-rsa-oaep-publ.pem y clave-rsa-oaep-priv.pem.

Si has recuperado la clave, vuelve a cifrarla con el mismo algoritmo. ¿Por qué son diferentes los textos cifrados?

Los textos cifrados son distintos debido a que cada vez que se vuelve a cifrar se añade un padding. En cada cifrado el padding cambia, haciendo que el cifrado sea totalmente distinto.

EJERCICIO 11. Nos debemos comunicar con una empresa, para lo cual, hemos decidido usar un algoritmo como el AES/GCM en la comunicación. Nuestro sistema, usa los siguientes datos en cada comunicación con el tercero:

Key:E2CFF885901B3449E9C448BA5B948A8C4EE322152B3F1ACFA0148FB3A426DB7 2 Nonce:9Yccn/f5nJJhAt2S

¿Qué estamos haciendo mal?

Los datos de comunicación que se emplean no son los correctos.
Es un cifrado con autentificación.
El nonce debe ser distinto, no se debe de reutilizar, en todos los mensajes para evitar que nos puedan hacer un ataque.