

# Informe de práctica final módulo **CRIPTOGRAFÍA**

---

JULIO DE 2022

REALIZADO POR

Angie Aristizabal Bernal

---



# Informe de práctica final

Este documento es un informe de la práctica final para el módulo de criptografía hecho por Angie Aristizabal Bernal para KeepCoding

En este resolveré los ejercicios y preguntas planteadas por el profesor sobre cada uno de los temas tocados a lo largo del módulo

He usado herramientas que iré mencionando en cada solución de cada punto. Principalmente Visual Studio Code para el escribir el código en el lenguaje de Python y KeyStore.

# 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?

## Solución

---

Para la solucionar esta incógnita simplemente recordamos la siguiente regla :  
 $K2 = K \oplus K1$  ;  $K1 = K \oplus K2$

Una vez teniendo en cuenta esto. Procedo a ir a Visual Studio Code para con el lenguaje Python buscar hacer el XOR que necesito

Lo primero que hago es importar una de las funciones utilizadas en clase para hacer la operación posteriormente :

```
2 def xor_data(binary_data_1, binary_data_2):
3     return bytes([b1^b2 for b1,b2 in zip (binary_data_1, binary_data_2)])
4
```

Después menciono tanto la clave fija en código como la clave final (en memoria) para pasarlas a bytes:

```
6 k2 = bytes.fromhex('A1EF2ABFE1AAEEFF')
7 k = bytes.fromhex('B1AA12BA21AABB12')
```

A continuación llamando la función importada previamente, pido que se muestre el resultado de la solución:

```
10
11 print(xor_data(k,k2).hex())
```

Con el código terminado, lo ejecuto, dándome como resultado la clave del Key Manager :

```
PS C:\Users\Angie\Desktop\KeepCoding\Criptografía> c:; cd 'c:\Users\Angie\Desktop\KeepCoding\Criptografía' & python 'c:\Users\Angie\Desktop\KeepCoding\Criptografía\Programs\Python\Python310\python.exe' 'c:\Users\Angie\.vscode\extensions\ms-python.python\python\debugpy\launcher' '49410' '--' 'c:\Users\Angie\Desktop\KeepCoding\Criptografía\10453805c00055ed'
```

Por lo tanto el valor ha puesto el Key Manager en properties para forzar dicha clave final es :

10453805c00055ed

Para corroborar esta información podemos hacer un  
10453805c00055ed XOR A1EF2ABFE1AAEEFF  
y podemos ver que el resultado efectivamente es : 10453805c00055ed

El archivo con el código escrito será adjuntado junto este informe.

Nombre del archivo : **ejercicio1.1.py**

---

-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?

## Solución

---

Para esto de nuevo recordamos la regla de la anterior incógnita. Así que si en esta ocasión la K1 es B98A15BA31AEBB22.

Se hace el mismo procedimiento. Un XOR entre ambos datos que tenemos.

Vuelvo a importar la función para la operación XOR usada durante el módulo:

```
2 def xor_data(binary_data_1, binary_data_2):
3     return bytes([b1^b2 for b1,b2 in zip (binary_data_1, binary_data_2)])
4
```

Pasamos como siempre la información a bytes:

```
5  # Primero paso la ambas claves a Bytes
6  k1 = bytes.fromhex('B98A15BA31AEBB22')
7  k2 = bytes.fromhex('A1EF2ABFE1AAEEFF')
8
```

Y pedimos que nos muestre el resultado del XOR entre ambos en formato hexadecimal y haciendo uso de la función escrita en un principio :

```
11  print(xor_data(k1,k2).hex())
```

Y al ejecutar el código tenemos nuestra respuesta a la pregunta :

```
PS C:\Users\Angie\Desktop\KeepCoding\Criptografía> c:; cd 'c:
Programs\Python\Python310\python.exe' 'c:\Users\Angie\.vscode\
..\debugpy\launcher' '49831' '--' 'c:\Users\Angie\Desktop\Kee
18653f05d00455dd
PS C:\Users\Angie\Desktop\KeepCoding\Criptografía> █
```

Entonces la clave será con la que se trabaje en memoria en este caso es :  
**18653f05d00455dd**

Y podemos corroborarlo ejerciendo la regla mencionada en un principio:  
 $K2 = K \wedge K1$  ;  $K1 = K \wedge K2$

El archivo con el código escrito será adjuntado junto este informe.

Nombre del archivo : **ejercicio1.2.py**

# 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:

ZCFJXRIFZABJ+GVWFRAAHIN2WV+G2P01IFRKEJICAGPOKPNZMIEXN3WXLGYX5WNNIOSY  
KFKNKG9GGSGGIAWAZG==

PARA ESTE CASO, SE HA USADO UN AES/CBC/PKCS7. SI LO DESCIFRAMOS, ¿QUÉ OBTENEMOS?

## Solución

### PASO 1

Como primer paso vamos a ir a Python para buscar la clave con la etiqueta mencionada.

Voy a usar parte de un código facilitado en el módulo por parte del profesor.

```
1 import jks
2 import os
3
4 path=os.path.dirname(__file__)
5 print(path)
6
7 keystore=path + "/KeyStorePracticas"
8 ks = jks.KeyStore.load(keystore, "123456")
9
10 for alias, sk in ks.secret_keys.items():
11     print("Secret key: %s" % sk.alias)
12     print(" Algorithm: %s" % sk.algorithm)
13     print(" Key size: %d bits" % sk.key_size)
14     print(" Key: %s" % "".join("{:02x}".format(b) for b in bytearray(sk.key)))
```

Primero importamos lo necesario para obtener nuestra clave

Después con la función path vamos a indicar la ruta que queremos que nuestro código nos ayude a hacer.

Escribimos la contraseña para ingresar al Key Store

Vamos a pedir los datos que queremos que nos muestre y como resultado obtendremos la clave con la etiqueta "cifrado-sim-aes-256" y el resto de información del resto de claves que tengamos en Key Store

```
Secret key: cifrado-sim-aes-256
 Algorithm: AES
 Key size: 256 bits
 Key: e2cff885901a5449e9c448ba5b948a8c4ee377152b3f1acfa0148fb3a426db72
PS C:\Users\Angie\Desktop\KeepCoding\Criptografía>
```

Ya tenemos nuestra clave para resolver cualquiera de las preguntas planteadas:

e2cff885901a5449e9c448ba5b948a8c4ee377152b3f1acfa0148fb3a426db72

## PASO 2

Para descifrar un código que tenemos en Base64, me voy a Visual Studio Code y trabajaré con el lenguaje de Python.  
Aquí primero importaré ciertas funciones para poder descifrar el texto :

```
1 import json
2 from Crypto.Cipher import AES
3 from Crypto.Util.Padding import pad, unpad
4 from Crypto.Random import get_random_bytes
5 from base64 import b64encode, b64decode
6 import base64
```

Después pasaré toda la información que necesito para descifrar el mensaje a bytes : la clave, el IV y obviamente el mismo texto

El IV podemos conocerlo sabiendo el tipo de algoritmo que estoy usando, un AES 256. Como sé que el IV debe ocupar lo que permite este algoritmo y sabiendo que 1 byte = 2 caracteres en hexadecimal , con todo esto puedo saber que el IV será de 32 "0" en total.

```
9 texto_cifrado_bytes=b64decode('zcFJxR1fzaBj+gVWFRAah1N2wv+G2P01ifrKejICaGpQkPnZMiexn3WXLGYX5WnNIosyKfkNKG9GGSgG1awaZg==')
10 clave = bytes.fromhex('e2cff885901a5449e9c448ba5b948a8c4ee377152b3f1acfa0148fb3a426db72')
11 iv_bytes = bytes.fromhex('00000000000000000000000000000000')
```

Una vez con todo en bytes, le indico las herramientas a usar para descifrar el texto: el IV y la clave ; con el modo AES/CBC que ya sabemos previamente que es con lo que estamos trabajando.

Después paso a la acción con la función de descifrar, dándole las herramientas y el mensaje a descifrar.

Una vez esto, le pido que me lo muestre tanto en hexadecimal como en texto como tal :

(También pido que si hay algún error me diga cual)

```
13
14 try:
15     cipher = AES.new(clave, AES.MODE_CBC, iv_bytes)
16     mensaje_des_bytes = unpad(cipher.decrypt(texto_cifrado_bytes), AES.block_size, style="pkcs7")
17     print("En hex: ", mensaje_des_bytes.hex())
18     print("El texto en claro es: ", mensaje_des_bytes.decode("utf-8"))
19
20 except (ValueError, KeyError) as error:
21     print('Problemas para descifrar....')
22     print("El motivo del error es: ", error)
```

Ejecutando todo esto, obtenemos el texto en hexadecimal y en texto claro, que es :

Esto es cifrado en bloque típico. Recuerda el padding.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

\Desktop\KeepCoding\Criptografía\TRABAJO FINAL\2\ejercicio2.1.py'

En hex: 4573746f20657320756e206369667261646f20656e20626c6f7175652074c3ad7069636f2e20526563756572646120656c2070616464696e672e

El texto en claro es: Esto es un cifrado en bloque típico. Recuerda el padding.

PS C:\Users\Angie\Desktop\KeepCoding\Criptografía\TRABAJO FINAL\2> |



## Pregunta 2

¿QUÉ OCURRE SI DECIDIMOS CAMBIAR EL PADDING A X923EN EL DESCIFRADO?

## Solución

Usando el código creado en el paso 2 intento cambiar el padding a la hora de descifrarlo y claramente me sale error ya que tiene que ser descifrado con el mismo estilo de padding que fue cifrado

```
\Desktop\KeepCoding\Criptografía\TRABAJO FINAL\2\ejercicio2.1.py'
Problemas para descifrar....
El motivo del error es: Unknown padding style
PS C:\Users\Angie\Desktop\KeepCoding\Criptografía\TRABAJO FINAL\2> |
```

## Pregunta 3

¿CUÁNTO PADDING SE HA AÑADIDO EN EL CIFRADO?

## Solución

Para esto voy a usar de nuevo el código creado en un principio y cambiaré una línea cómo si supusiera que no existe padding en este texto cifrado. Esto hará que la respuesta en hexadecimal no elimine el padding, y me lo enseñe:

```
13 try:
14     cipher = AES.new(clave, AES.MODE_CBC, iv_bytes)
15     #mensaje_des_bytes = unpad(cipher.decrypt(texto_cifrado_bytes), AES.block_size, style="pkcs7")
16     mensaje_des_bytes = cipher.decrypt(texto_cifrado_bytes)
17     print("En hex: ", mensaje_des_bytes.hex())
18     print("El texto en claro es: ", mensaje_des_bytes.decode("utf-8"))
19
```

Así la respuesta que recibo es la siguiente :

```
\Desktop\KeepCoding\Criptografía\TRABAJO FINAL\2\ejercicio2.1.py'
En hex: 4573746f20657320756e2063696667261646f20656e20626c6f7175652074c3ad7069636f2e20526563756572646120656c2070616464696e672e060606060606
El texto en claro es: Esto es un cifrado en bloque típico. Recuerda el padding.♦♦♦♦♦♦
PS C:\Users\Angie\Desktop\KeepCoding\Criptografía\TRABAJO FINAL\2> |
```

Así que el padding es de 6 bytes, 12 caracteres en hexadecimal

El archivo con el código escrito de este ejercicio será adjuntado junto este informe.

Nombre del archivo :

[ejercicio2.py](#) (para el código donde saco la clave de KeyStore)

[ejercicio2.1.py](#)



# 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".

¿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.

## Solución

Para mejorar el sistema y garantizar la confidencialidad y la integridad del mensaje haré uso del Poly 1305. Lo cual nos ayuda a verificar que ningún dato ha sido modificado.

Una vez decidido esto, empiezo con el código, dónde lo primero que haré será pasar todas las funciones que necesite :

```
1 from Crypto.Cipher import ChaCha20_Poly1305
2 from base64 import b64decode, b64encode
3 from Crypto.Random import get_random_bytes
4 import json
5
```

Después como siempre necesito pasar todos los datos a bytes:

```
6 try:
7
8     textoPlano_bytes = bytes('Este curso es de lo mejor que podemos encontrar en el mercado', 'UTF-8')
9     clave_bytes = bytes.fromhex('979DF30474898787A45605CCB9B936D33B780D03CABC81719D52383480DC3120')
10    nonce_mensaje_bytes = b64decode('9Yccn/f5nJJhAt2S')
11    datos_asociados_bytes = bytes('Datos no cifrados sólo autenticados', 'utf-8')
12    cipher = ChaCha20_Poly1305.new(key=clave_bytes, nonce=nonce_mensaje_bytes)
13    cipher.update(datos_asociados_bytes)
14    texto_cifrado_bytes, tag_bytes = cipher.encrypt_and_digest(textoPlano_bytes)
15    mensaje_enviado = { "nonce": b64encode(nonce_mensaje_bytes).decode(), "datos asociados": b64encode(datos_asociados_bytes).decode(), "texto cifrado":
16    b64encode(texto_cifrado_bytes).decode(), "tag": b64encode(tag_bytes).decode()}
17    json_mensaje = json.dumps(mensaje_enviado)
18    print("Mensaje: ", json_mensaje)
19
20 except (ValueError, KeyError) as error:
21     print("Problemas al cifrar....")
22     print("El motivo del error es: ", error)
```

La línea nº15 dice :

```
mensaje_enviado = { "nonce": b64encode(nonce_mensaje_bytes).decode(), "datos asociados":
b64encode(datos_asociados_bytes).decode(), "texto cifrado":
b64encode(texto_cifrado_bytes).decode(), "tag": b64encode(tag_bytes).decode() }
```

Después de pasar todo a bytes, he creado unos datos asociados que servirán para que el receptor pueda verificar que el mensaje no ha sido alterado y asegurar la integridad del mismo.

Hago uso del cipher que me ayudará a realizar la acción de cifrar. Dándole lo que necesita para llevarlo a cabo, como la clave y el nonce

Después añadido a esto los datos asociados mencionados previamente

A continuación cifro el texto y creo el tag al mismo tiempo.

Una vez todo esto hecho, creo un mensaje json que servirá para enseñarle al remitente todo los datos necesarios para descifrar y verificar la integridad/confidencialidad del mensaje

Y pido que si llega a existir un error con el código al cifrar me diga cual es .

Con esto, recibo la siguiente respuesta al ejecutarlo :

```
Mensaje: {"nonce": "9Yccn/f5nJJhAt2S", "datos asociados":  
"RGF0b3Mgbm8gY2lmcmFkb3Mgc8OzbG8gYXV0ZW50aWNhZG9z", "texto  
cifrado":  
"EWGrU8t3Mr7pqLjrFz1hGgMkqhxZBXsbqK8nA1gqmkcMXkqzA5OdUsPYPO8as  
Rk1nZo30zieQ56ggemkUQ==", "tag": "hDNXqvnIV89m6CIF4KLzWA=="}
```

El archivo con el código escrito de este ejercicio será adjuntado junto este informe.

Nombre del archivo :  
**ejercicio3.py**

# Ejercicio 4

TENEMOS EL SIGUIENTE JWT, CUYA CLAVE ES "KEEPCODING".

EYJ0EXAIOIJKV1QILCJHBGCIOIJIUZI1NIJ9.EYJ1C3VHCMLVIJOIRMVSAXBLIFJVZHJCDT  
AWZWRNDWV6I1WICM9SIJOIAXNOB3JTYWWIFQ.-  
KIAA8CJKAMJWRUINVHGGEJU8K2WIERDXQP\_IFXUMM8

1.¿QUÉ ALGORITMO DE FIRMA HEMOS REALIZADO?

## Solución

---

Para saber que algoritmo se ha usado para la firma he hecho uso de una pagina web : [jwt.io](https://jwt.io)

Esta página fue facilitada por el profesor

Donde al introducir el JWT junto a la contraseña dada en el enunciado del ejercicio recibimos el tipo de y el algoritmo usado. Que es en este caso un HMAC SHA 256



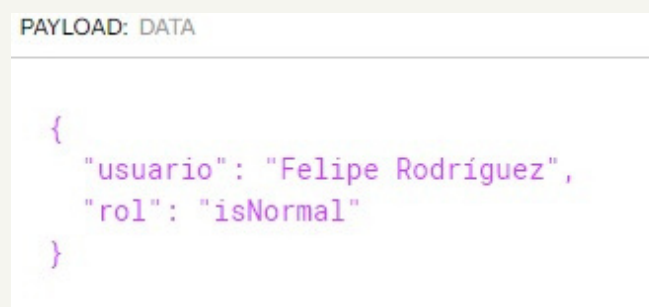
2 . ¿CUÁL ES EL BODY DEL JWT?

## Solución

---

De nuevo haciendo uso de la herramienta de la primera pregunta podemos ver el body del JWT .

El cual es el impreso en esta imagen



**3 . UN HACKER ESTÁ ENVIANDO A NUESTRO SISTEMA EL SIGUIENTE JWT:**  
**EYJ0EXAIOIJKV1QILCJHBGCI0IJIUZI1NIJ9.EYJ1C3VHCMLVIJOIRMVSA XBLIFJVZHJCDT**  
**AWZWRNDWV6I IWICM9SIJOIAXNBZG1PBIJ9.-**  
**KIAA8CJ KAMJWRUINVHGGEJU8K2WIERDXQP\_IFXUMM8**

**¿QUÉ ESTÁ INTENTANDO REALIZAR?**

## Solución

---

Para solucionar esta incógnita podemos comparar los resultados del JWT dado en un principio con los resultados del JWT enviado por el hacker. Hago uso de nuevo de la página [jwt.io](https://jwt.io) y comparo los resultados.

### Original

HEADER: ALGORITHM & TOKEN TYPE
<pre>{   "typ": "JWT",   "alg": "HS256" }</pre>
PAYLOAD: DATA
<pre>{   "usuario": "Felipe Rodríguez",   "rol": "isNormal" }</pre>

### Hacker

HEADER: ALGORITHM & TOKEN TYPE
<pre>{   "typ": "JWT",   "alg": "HS256" }</pre>
PAYLOAD: DATA
<pre>{   "usuario": "Felipe Rodríguez",   "rol": "isAdmin" }</pre>

Como podemos observar, la diferencia es que esta intentando burlar el sistema y coger privilegios como administrador

**4.¿QUÉ OCURRE SI INTENTAMOS VALIDARLO CON PYJWT?**

## Solución

---

Para esto hago uso de Visual Studio Code y usare el lenguaje Python. Debo en primer lugar importar JWT. Y a continuación haré uso del "jwt.decode"

En el `jwt.decode`, introduzco el `jwt`, la clave y el algoritmo que se esta usando. Al ejecutarlo nos sale este error porque el atacante no tiene nuestra firma por lo tanto es invalida:

```
1 import jwt
2
3
4 decode_jwt = jwt.decode ('eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmlvIjoiriRmVsaXB1IFJvZHLDrWd1ZXoILCJyb2wiOiJpc05vcm1hbCJ9.72TWZzxe3LTSHqMosrT-R08y_NqTeECRQ', "KeepCoding", algorithms="HS256")

Exception has occurred: InvalidSignatureError ×
Signature verification failed

File "C:\Users\Angie\Desktop\KeepCoding\Criptografía\TRABAJO FINAL\4\pyjwt.py", line 7, in <module>
    decode_jwt = jwt.decode
('eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmlvIjoiriRmVsaXB1IFJvZHLDrWd1ZXoILCJyb2wiOiJpc05vcm1hbCJ9.72TWZzxe3LTSHqMosrT-R08y_NqTeECRQ', "KeepCoding", algorithms="HS256")

5
6
7 print(decode_jwt)
```

El archivo con el código escrito de este ejercicio será adjuntado junto este informe.

Nombre del archivo :  
**ejercicio4.py**

# Ejercicio 5

EL SIGUIENTE HASH SE CORRESPONDE CON UN SHA3 KECCAK DEL TEXTO "EN KEEPCODING APRENDEMOS CÓMO PROTEGERNOS CON CRIPTOGRAFÍA".

BCED1BE95FBD85D2FFCCE9C85434D79AA26F24CE82FBD4439517EA3F072D56FE

1. ¿QUÉ TIPO DE SHA3 HEMOS GENERADO?

## Solución

---

Estamos hablando de un SHA3 256, cosa que podemos saber gracias a la longitud de este . Ya que tiene 64 caracteres en hexadecimal.

---

Y SI HACEMOS UN SHA2, Y OBTENEMOS EL SIGUIENTE RESULTADO:

4CEC5A9F85DCC5C4C6CCB603D124CF1CDC6DFE836459551A1044F4F2908AA5D63739506  
F646883 3D77C07CFD69C488823B8D858283F1D05877120E8C5351C833

2. ¿QUÉ HASH HEMOS REALIZADO?

## Solución

---

Hemos realizado un SHA2 DE 512 bits (64 bytes) ya que por la longitud de este podemos saberlo.

---

3. 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?

## Solución

---

Hago uso del hashlib y uso de guía el ejemplo del profesor para realizar un hash :



```

1  import hashlib
2
3
4  s = hashlib.sha3_256()
5
6  s.update(bytes("En KeepCoding aprendemos cómo protegernos con criptografía.", "UTF-8"))
7
8  print(s.hexdigest())

```

Y así obtengo el hash :

302be507113222694d8c63f9813727a85fef61a152176ca90edf1cfb952b19bf

Destaco el ver el hecho de cambiar un sólo carácter nos sirve para recibir un hash completamente diferente

[El archivo con el código escrito de este ejercicio será adjuntado junto este informe.](#)

Nombre del archivo :

[ejercicio5.py](#)

## Ejercicio 6

**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.**

## Solución

Para calcular el HMAC usare una vez mas Visual Studio Code, donde importare lo necesario. Pasaré toda la información a bytes como siempre y haré uso de la función importada en un principio.

Esta función genera el HMAC con los datos facilitados en bytes así que al ejecutar el código con un "print" podré ver el HMAC generado

```

1  from Crypto.Hash import HMAC, SHA256
2
3  secret = bytes.fromhex('2712A51C997E14B4DF08D55967641B0677CA31E049E672A4B06861AA4D5826EB')
4  msg0= bytes('Siempre existe más de una forma de hacerlo, y más de una solución válida.', 'utf-8')
5
6  h = HMAC.new(secret, msg=msg0, digestmod=SHA256)
7  print(h.hexdigest())
8

```

Y el HMAC obtenido es :

d0b686743822793fb185263f1fa4ded09bd557bcd341ac53e06dad76238c8e09

El archivo con el código escrito de este ejercicio será adjuntado junto este informe.

Nombre del archivo :  
[ejercicio6.py](#)

## 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?**

### Solución

---

Porque el Hash SHA-1 es un tipo de hash que esta roto a día de hoy. En 2017 encontraron ataques de colisión.

---

**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?**

### Solución

---

Necesitamos garantizar la integridad así que yo propondría hacer uso de un MAC donde ingresaremos los datos de los trabajadores o clientes y la información delicada igualmente cifrada. Para asegurarnos de que aún con el MAC no se ponga en peligro información valiosa.

---

**PARECE QUE EL RESPONSABLE SE HA QUEDADO CONFORME, TRAS MEJORAR LA PROPUESTA DEL SHA 256, NO OBSTANTE, HAY MARGEN DE MEJORA. ¿QUÉ PROPONDRÍAS?**

### Solución

---

Integraría algún sistema de verificación como son los OTPs o SSO para así asegurarnos de que cada persona que ingrese sea legítima . También podría hacer uso de autorización por medio de JWT

# Ejercicio 8

TENEMOS LA SIGUIENTE API REST, MUY SIMPLE.

REQUEST:

POST /MOVIMIENTOS

C a m p o	Tipo	Requiere confidencialidad	Observaciones
id Usuario	Number	N	Identificador
Usuario	String	S	Nombre y Apellidos
Tarjeta	Number	S	

TPETICIÓN DE EJEMPLO QUE SE DESEA ENVIAR:

`{"IDUSUARIO":1,"USUARIO":"JOSÉ MANUEL BARRIO  
BARRIO","TARJETA":4231212345676891}` RESPONSE:



C a m p o	Tipo	Requiere confidencialidad	Observaciones
id Usuario	Number	N	Identificador
movTarjeta	Array	S	Formato de 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?**

**(IMPORTANTE, DE CARA A CALCULAR CUALQUIER COSA, SE DEBEN QUITAR RETORNOS DE CARRO, ESPACIOS INTERMEDIOS, USAR LAS MISMAS COMILLAS, ETC)**

## Solución

---

Yo optaría por toda información importante cifrarlo y usar también HMAC para así trabajar en la verificación. Después de esto, metería toda la información en un JSON, con una firma y así si se llega a modificar el mensaje tendremos forma de saberlo . Trabajar con el método Diffie-Hellman para el intercambio de claves también es una buena idea.

# 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.**

## Solución

---

Para este ejercicio usare la herramienta facilitada en clase : GNUPG . Lo que me llevará a solucionar todo por medio de la consola

Primero importaré todos los datos proporcionados en la página de GitHub (que ya han sido previamente puestos en archivos .txt) dentro del programa con el siguiente comando:

```
gpg --import (nombre del archivo)
```

Una vez todo importado he usado el siguiente comando para verificar el archivo `MensajeRespoDeRaulARRHH.txt.sig` :

```
gpg -- verify (nombre del archivo )
```

Recibiendo la siguiente respuesta:

```
C:\Users\Angie\Desktop\KeepCoding\Criptografía\TRABAJO FINAL\9
λ gpg --verify MensajeRespoDeRaulARRHH.txt.sig
gpg: Firmado el 26/06/2022 01:47:01 p. m. Hora de verano romance
gpg: usando EDDSA clave 1BDE635E4EAE6E68DFAD2F7CD730BE196E466101
gpg: emisor "pedro.pedrito.pedro@empresa.com"
gpg: Firma correcta de "Pedro Pedrito Pedro <pedro.pedrito.pedro@empresa.com>" [desconocido]
gpg: WARNING: The key's User ID is not certified with a trusted signature!
gpg: No hay indicios de que la firma pertenezca al propietario.
Huellas dactilares de la clave primaria: 1BDE 635E 4EAE 6E68 DFAD 2F7C D730 BE19 6E46 6101
gpg: ADVERTENCIA: firma no separada; ¡el archivo MensajeRespoDeRaulARRHH.txt NO ha sido verificado!
```



Y aquí como se ve en la imagen, confirmo que el mensaje ha sido enviado con la firma correcta.

Ya que el Warning es porque no tiene un certificado como tal. Ya que esto es para un ejercicio y aprender.

---

**ASÍ MISMO, SE REQUIERE FIRMAR EL SIGUIENTE MENSAJE CON LA CLAVE CORRESPONDIENTE DE LAS ANTERIORES, SIMULANDO QUE ERES PERSONAL DE RRHH.**

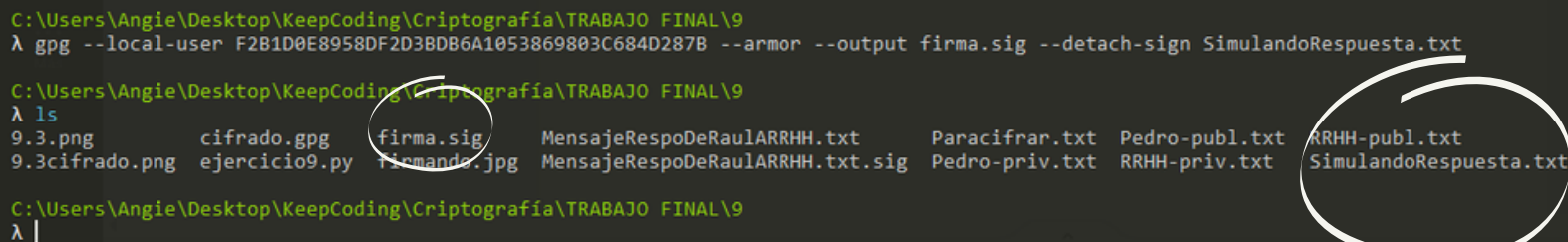
VIENDO SU PERFIL EN EL MERCADO, HEMOS DECIDIDO ASCENDERLE Y MEJORARLE UN 25% SU SALARIO. SALUDOS.

## Solución

---

Lo primero es pasar el mensaje a firmar a un archivo .txt. Después de esto

```
gpg --local-user F2B1D0E8958DF2D3BDB6A1053869803C684D287B --armor --output firma.sig --detach-sign SimulandoRespuesta.txt
```



The screenshot shows a terminal window with the following content:

```
C:\Users\Angie\Desktop\KeepCoding\Criptografía\TRABAJO FINAL\9
λ gpg --local-user F2B1D0E8958DF2D3BDB6A1053869803C684D287B --armor --output firma.sig --detach-sign SimulandoRespuesta.txt

C:\Users\Angie\Desktop\KeepCoding\Criptografía\TRABAJO FINAL\9
λ ls
9.3.png          cifrado.gpg      firma.sig        MensajeRespoDeRaulARRHH.txt  Paracifrar.txt  Pedro-publ.txt  RRHH-publ.txt
9.3cifrado.png  ejercicio9.py    firmando.jpg     MensajeRespoDeRaulARRHH.txt.sig  Pedro-priv.txt  RRHH-priv.txt  SimulandoRespuesta.txt

C:\Users\Angie\Desktop\KeepCoding\Criptografía\TRABAJO FINAL\9
λ |
```

Two white circles are drawn on the image: one around 'firma.sig' and another around 'SimulandoRespuesta.txt'.

A continuación he lanzado un ls para confirmar la creación de la firma y así como se ve en la imagen se encuentra la firma y el texto a firmar

---

**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.

## Solución

---

Primero he escrito el mensaje en un archivo .txt

Dentro de la misma consola voy a ejecutar el siguiente comando:

```
λ gpg --output cifrado.gpg --encrypt --recipient
F2B1D0E8958DF2D3BDB6A1053869803C684D287B --recipient
1BDE635E4EAE6E68DFAD2F7CD730BE196E466101 --armor Paracifrar.txt
```

```

C:\Users\Angie\Desktop\KeepCoding\Criptografia\TRABAJO FINAL\9
λ gpg --output cifrado.gpg --encrypt --recipient F2B1D0E8958DF2D38D86A1053869803C684D287B --recipient 1BDE635E4EAE6E68DFAD2F7CD730BE196E466101 --armor Paracifrar.
gpg: 25D6D0294035B650: No hay seguridad de que esta clave pertenezca realmente
al usuario que se nombra

sub cv25519/25D6D0294035B650 2022-06-26 Pedro Pedrito Pedro <pedro.pedrito.pedro@empresa.es>
Huella clave primaria: 1BDE 635E 4EAE 6E68 DFAD 2F7C D730 BE19 6E46 6101
Huella de subclave: 8E8C 6669 AC44 3271 428C C244 2506 D029 4035 B650

No es seguro que la clave pertenezca a la persona que se nombra en el
identificador de usuario. Si *realmente* sabe lo que está haciendo,
puede contestar sí a la siguiente pregunta.

¿Usar esta clave de todas formas? (s/N) s
gpg: 7C1A46EA20B0546F: No hay seguridad de que esta clave pertenezca realmente
al usuario que se nombra

sub cv25519/7C1A46EA20B0546F 2022-06-26 RRHH <RRHH@RRHH>
Huella clave primaria: F2B1 D0E8 958D F2D3 BDB6 A105 3869 803C 684D 287B
Huella de subclave: 811D 89A3 6199 A7C9 0BFE 69D6 7C1A 46EA 20B0 546F

No es seguro que la clave pertenezca a la persona que se nombra en el
identificador de usuario. Si *realmente* sabe lo que está haciendo,
puede contestar sí a la siguiente pregunta.

¿Usar esta clave de todas formas? (s/N) s

C:\Users\Angie\Desktop\KeepCoding\Criptografia\TRABAJO FINAL\9
λ ls
9.3.png cifrado.gpg Firma.jpg.png firmando.jpg MensajeRespoDeRaulARRHH.txt.sig Pedro-priv.txt RRHH-priv.txt SimulandoRespuesta.txt
9.3cifrado.png ejercicio9.py firma.sig MensajeRespoDeRaulARRHH.txt Paracifrar.txt Pedro-publ.txt RRHH-publ.txt verificación.jpg
C:\Users\Angie\Desktop\KeepCoding\Criptografia\TRABAJO FINAL\9

```

Contesto que si quiero usar esa claves de todas formas. He escrito el fingerprint de RRHH y el de Pedro  
Después de esto pido que me muestre el archivo gpg creado , con un cat cifrado.gpg

```

9.3.png cifrado.gpg Firma.jpg.png firmando.jpg
9.3cifrado.png ejercicio9.py firma.sig MensajeRespoDeRaulARRHH.txt

C:\Users\Angie\Desktop\KeepCoding\Criptografia\TRABAJO FINAL\9
λ cat cifrado.gpg
-----BEGIN PGP MESSAGE-----

hF4DfBpG6iCwVG8SAQdAH/eScBPW+7W4deWfm4wvQm6up2wvQVhXRGZ5btB/uE8w
KgY+mgNTlI1I8uz7Aex5ysBfma4gZiElm/E1BibJCGokzffV0zXCQE1kY1nL7mEo
hF4DJdbQKUA1t1lASAQdAQCPzg0Yrb0ItBJmCYZA3Pr86Vnsye3AG7fFbnZeObkgw
Ws6mG280ecnka3sgSJskTQX9uH4reSx//Pg4jlCBiZTKuh8Dt0PCIABRH/QAL2GC
1IwBCQIQ4JbojF/Hy/SHehVgfuwTSu+BUzeapilMvYmk7CRUMtxps32qJY1pqZZs
QGLimwFrCAMau3fxx1MoCuPyavnOWqdX+pjfbP7vPwu0c+1KJ5LitrOMNDqNHbMK
fmBSK0lgCZuGwMnW03kg3gJeg8kyvbd4+FDqg5TxxS2TzVwC0rJS9eX/QDtLhg==
=heIN
-----END PGP MESSAGE-----

C:\Users\Angie\Desktop\KeepCoding\Criptografia\TRABAJO FINAL\9
λ

```

El archivo con lo escrito de este ejercicio será adjuntado junto este informe.

Nombre del archivo :  
**SimulandoRespuesta.txt**  
**firma.sig**  
**cifrado.gpg**

# Ejercicio 10

NUESTRA COMPAÑÍA TIENE UN CONTRATO CON UNA EMPRESA QUE NOS DA UN SERVICIO DE ALMACENAMIENTO DE INFORMACIÓN DE VIDEO LLAMADAS. PARA LO CUAL, LA MISMA NOS ENVÍA LA CLAVE SIMÉTRICA DE CADA VIDEO LLAMADA 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  
6B6016CD32E816336CAD7784989AF89FF853A3ACD796813EAD65CA3A10BBF58C621  
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?

## Solución

Para este punto he usado Visual Studio Code, así que por medio del siguiente código donde primero indico de donde leer las claves.

A continuación importo estas.

Hago uso del de PKCS1\_OAEP a la hora de descifrar , e ingresando el texto cifrado que nos dan en el enunciado , lo descifro y pido que lo muestre de manera hexadecimal.

```
ejercicio10.py
ejercicio10.py > ...
1  import os
2  from Crypto.PublicKey import RSA
3  from Crypto.Hash import SHA256
4  from Crypto.Cipher import PKCS1_OAEP
5
6  my_path = os.path.dirname(__file__)
7  path_file_publ = my_path + "/clave-rsa-oaep-publ.pem"
8  path_file_priv = my_path + "/clave-rsa-oaep-priv.pem"
9
10 keypub = RSA.import_key(open(path_file_publ).read())
11 keypriv = RSA.import_key(open(path_file_priv).read())
12
13
14 decryptor = PKCS1_OAEP.new(keypriv, SHA256)
15 textocifrado = bytes.fromhex('7edee3ec0b808c440078d63ee65b17e85f0c1adbc0da1b7fa842f24fb06b332c156038062d9daa8ccfe83bace1dca')
16 decrypted = decryptor.decrypt(textocifrado)
17 print('Descifrado:', decrypted.hex())
18
19 cipher = PKCS1_OAEP.new(keypub, SHA256)
20 encrypted = cipher.encrypt(decrypted)
21 print('Vuelto a cifrar:', encrypted.hex())
22
```

Después de descifrarlo he hecho el mismo procedimiento pero a la inversa. Cifrándolo.

Y como resultado de este código he recibido la respuesta al ejercicio :

```
22
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
; & 'C:\Users\Angie\AppData\Local\Programs\Python\Python310\python.exe' 'c:\Users\Angie\.vscode\extensions\ms-python.python-2022.10.1\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '57895' '--' 'c:\Users\Angie\Desktop\KeepCoding\Criptografía\TRABAJO FINAL\10\ejercicio10.py'
Descifrado: e2cff885901a5449e9c448ba5b948a8c4ee377152b3f1acfa0148fb3a426db72
Vuelto a cifrar 543c54cf3f2db66189bf48989c16a0c951d82e6559f2fc83199162f9482378898b42b64a57cb7841c5ab0d52eaa109acbc3c75b0e6435fa6d6d1fbf4582d09f9e44b4754fbf97f3dc662106f02c9d0bfd4085efe32cd26f255971c53f1c7bcd9a50e70bd14628f7ffbe8b878c38bb4893e218b7ae18d3c23b9f21b29712b189d43c522c8e031250593198e8520ff029c84afe9291426a0d9664252ada0463853fb8e1429ad38516240c05e8d9fb6929e663d708ca4ecb3f9e30b376dc5e8833bd82e17b2ff8d8d3abc68dcf5951fd9206838b4acc10d6eb0884061f43f7c6d9d0c2ecd60b6a64eca0ecc328f80c4de0f7588300243b9357b5ee8ddc6edc6bd98c
PS C:\Users\Angie\Desktop\KeepCoding\Criptografía\TRABAJO FINAL\10> |
```

En base a esto , puedo ver que obviamente son distintos cifrados y es gracias a una de las propiedades de OAEP. Dónde añade una especie de código propio aleatorio al principio que va cambiando. Si yo ejecutara de nuevo este mismo código el resultado cambiaría de nuevo, como lo muestra la siguiente imagen :

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
; & 'C:\Users\Angie\AppData\Local\Programs\Python\Python310\python.exe' 'c:\Users\Angie\.vscode\extensions\ms-python.python-2022.10.1\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '57940' '--' 'c:\Users\Angie\Desktop\KeepCoding\Criptografía\TRABAJO FINAL\10\ejercicio10.py'
Descifrado: e2cff885901a5449e9c448ba5b948a8c4ee377152b3f1acfa0148fb3a426db72
Vuelto a cifrar: bcb3559be1e92efa3ce9a0c94f7420eeceb48544892b2c04ff073631060f5cbfb3c1d98bac368069ebd23fda27ddbba66b38366dd02b7204c268cc40dc7824fbb7f164e4f26cee1abc47d81a2692447126944b634c2e7734c4f9fc83ae12ec329ef0db133bbe46e071be40037505c5bb4bc59a97cfa21aa29a69185ffab65da90ecc4d803c46d04811fb965efe3d932a08f497513e37b67e4e63011a009fa6914e2c60aae1ecf1e7b0e9fc56e4b7cf796e163575f04ca42e0ab6e790548c323ce806b13b08b4c65338e911fcae248dfdc59054fc37835b6e29f8a2e913fbab7d58298794ca8c1327d2536e38964fc984a6756ec332a54e9c1d8cdcc78ed2c495
PS C:\Users\Angie\Desktop\KeepCoding\Criptografía\TRABAJO FINAL\10> |
```

El archivo con lo escrito de este ejercicio será adjuntado junto este informe.

Nombre del archivo :

**ejercicio10.py**

# 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:E2CFF885901B3449E9C448BA5B948A8C4EE322152B3F1ACFA0148FB3A426DB72**  
**NONCE:9YCCN/F5NJJHAT2S**

**¿QUÉ ESTAMOS HACIENDO MAL?**

## Solución

---

Si este nonce que me dan es por lo que parece fijo, para cada comunicación que hay, es un error grande. Ya que el nonce debe ir cambiando cada vez con cada mensaje que se envía así que no debería estar entre la información que se usa porque debería no ser uno fijo sino ir cambiando y no dejar ninguna cuerda de la que tirar a ningún atacante

FIN DEL INFORME DE LA PRÁCTICA DE CRIPTOGRAFÍA  
JULIO 2022