



PRÁCTICA CRIPTOGRAFÍA



22 DE DICIEMBRE DE 2024
ANDRES JESUS RICAURTE VALERA
KEEPCODING

Ejercicio 1.

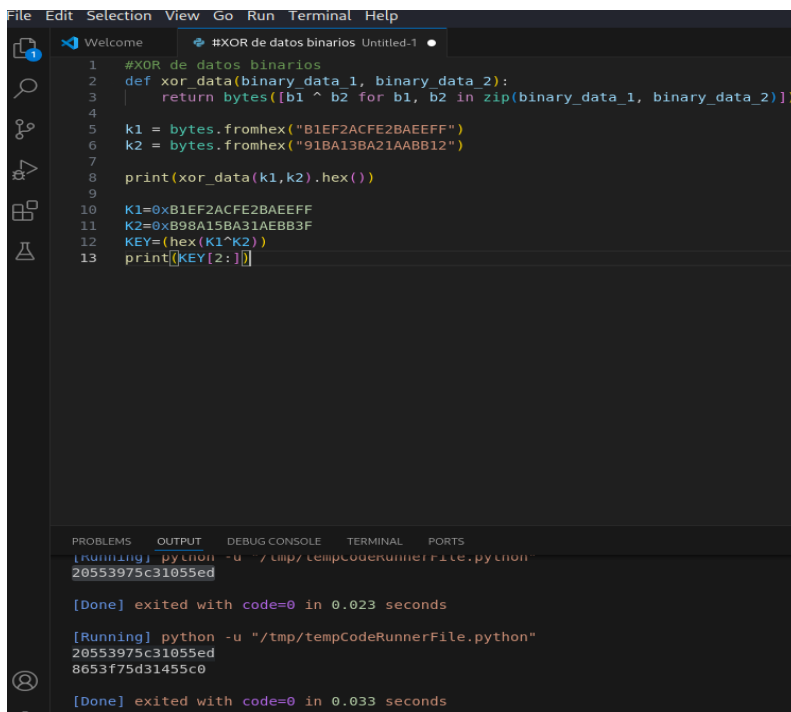
Tenemos un sistema que usa claves de 16 bytes. Por razones de seguridad vamos a proteger la clave de tal forma que ninguna persona tenga acceso directamente a la clave. Por ello, vamos a realizar un proceso de disociación de la misma, en el cuál tendremos, una clave fija en código, la cual, sólo el desarrollador 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 B1EF2ACFE2BAEEFF, mientras que en desarrollo sabemos que la clave final (en memoria) es 91BA13BA21AABB12. ¿Qué valor ha puesto el Key Manager en properties para forzar dicha clave final?

20553975c31055ed

La clave fija, recordemos es B1EF2ACFE2BAEEFF, mientras que en producción sabemos que la parte dinámica que se modifica en los ficheros de propiedades es B98A15BA31AEBB3F. ¿Qué clave será con la que se trabaje en memoria?

08653F75D31455C0



```
File Edit Selection View Go Run Terminal Help
Welcome #XOR de datos binarios Untitled-1
1 #XOR de datos binarios
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
5 k1 = bytes.fromhex("B1EF2ACFE2BAEEFF")
6 k2 = bytes.fromhex("91BA13BA21AABB12")
7
8 print(xor_data(k1,k2).hex())
9
10 K1=0xB1EF2ACFE2BAEEFF
11 K2=0xB98A15BA31AEBB3F
12 KEY=(hex(K1^K2))
13 print(KEY[2:])

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
[Running] python -u "/tmp/tempCodeRunnerFile.py"
20553975c31055ed

[Done] exited with code=0 in 0.023 seconds

[Running] python -u "/tmp/tempCodeRunnerFile.py"
20553975c31055ed
08653f75d31455c0

[Done] exited with code=0 in 0.033 seconds
```

Ejercicio 2.

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

TQ9SOMKc6aFS9SlxhfK9wT18UXpPCd505Xf5J/5nLI7Of/o0QKIWXg3nu1RRz4QWElezdrLAD5LO4US
t3aB/i50nvvJbBiG+le1ZhpR84ol=

Para este caso, se ha usado un AES/CBC/PKCS7. Si lo desciframos, ¿qué obtenemos?

Texto descifrado en claro: Esto es un cifrado en bloque típico. Recuerda, vas por el buen camino. Ánimo.

¿Qué ocurre si decidimos cambiar el padding a x923 en el descifrado? ¿Cuánto padding se ha añadido en el cifrado?

Genera el mismo resultado.

```
19
20 # Descifrado
21 cipher = AES.new(key, AES.MODE_CBC, iv)
22 decrypted = unpad(cipher.decrypt(encrypted_bytes), AES.block_size, style="pkcs7")
23
24
25 print("Texto descifrado en claro:", decrypted.decode('utf-8'))
26
27 #Descifrado con x923
28
29 cipher = AES.new(key, AES.MODE_CBC, iv)
30 decrypted = unpad(cipher.decrypt(encrypted_bytes), AES.block_size, style="x923")
31
32
33
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

[Running] python -u "/tmp/tempCodeRunnerFile.python"

Texto descifrado en claro: Esto es un cifrado en bloque típico. Recuerda, vas por el buen camino. Ánimo.

[Done] exited with code=0 in 0.089 seconds

[Running] python -u "/tmp/tempCodeRunnerFile.python"

Texto descifrado en claro: Esto es un cifrado en bloque típico. Recuerda, vas por el buen camino. Ánimo.

[Done] exited with code=0 in 0.089 seconds

Se requiere cifrar el texto “KeepCoding te enseña a codificar y a cifrar”. La clave para ello, tiene la etiqueta en el Keystore “cifrado-sim-chacha20-256”. El nonce “9Yccn/f5nJJhAt2S”. El algoritmo que se debe usar es un Chacha20.

¿Cómo podríamos mejorar de forma sencilla el sistema, de tal forma, que no sólo garanticemos la confidencialidad sino, además, la integridad del mismo? Se requiere obtener el dato cifrado, demuestra, tu propuesta por código, así como añadir los datos necesarios para evaluar tu propuesta de mejora.

Solución de ejercicio mostrada en GitHub.

Ejercicio 4.

Tenemos el siguiente jwt, cuya clave es “Con KeepCoding aprendemos”.

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmIvIjoiriRG9uIFBlcGl0byBkZSBsb3MgcGFsb3RlcyIsInJvbmCI6ImIzTm9ybWFSliwiaWF0IjoxNjY3OTMzNTMzfQ.gfhw0dDxp6oixMLXXRP97W4TDTrv0y7B5YjD0U8ixrE
```

¿Qué algoritmo de firma hemos realizado?

HS256

The screenshot shows a web-based JWT decoder interface. At the top, there's a green banner with the text "Last build: 2 months ago - Version 10 is here! Read about the new features here". Below this, the interface is divided into two main sections: "Recipe" and "Input". The "Recipe" section on the left has a green background and contains a dropdown menu set to "Alphabet A-Za-z0-9+/", two checkboxes labeled "Remove non-alphabet chars" and "Strict mode" (both checked), and some navigation icons. The "Input" section on the right has a light gray background and contains the JWT string: `eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmIvIjoiriRG9uIFBlcGl0byBkZSBsb3MgcGFsb3RlcyIsInJvbmCI6ImIzTm9ybWFSliwiaWF0IjoxNjY3OTMzNTMzfQ.gfhw0dDxp6oixMLXXRP97W4TDTrv0y7B5YjD0U8ixrE`. Below the input, there's a section labeled "Output" with a light gray background, displaying the decoded JSON: `{ "typ": "JWT", "alg": "HS256" }`. At the bottom of the interface, there's a small status bar showing "abc 36" and "1".

¿Cuál es el body del jwt?

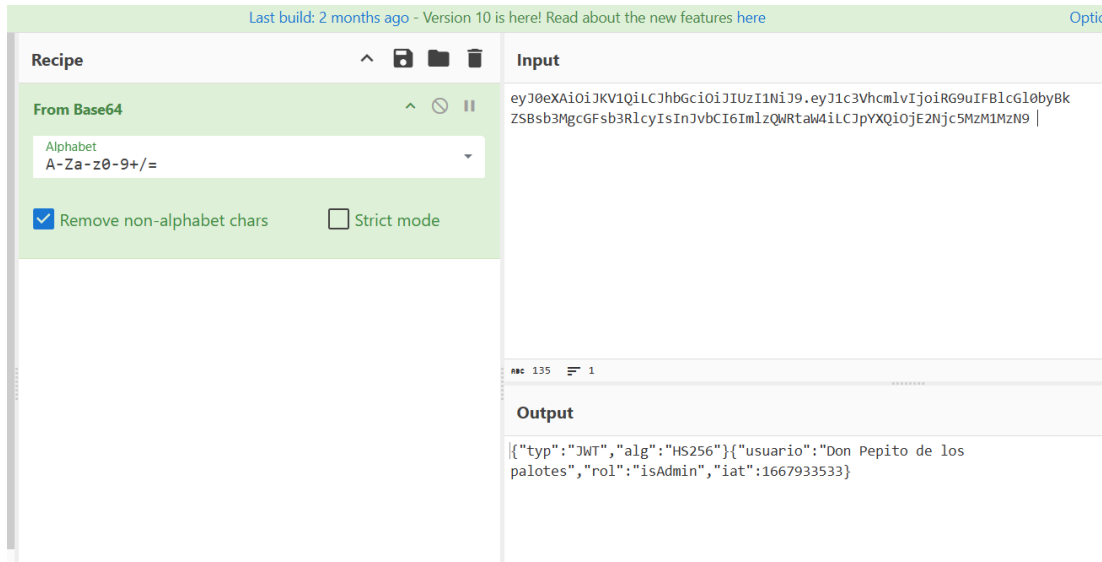
This screenshot shows the same JWT decoder tool as the previous one, but with a different input JWT. The "Recipe" section on the left remains the same, with the "Alphabet" dropdown set to "A-Za-z0-9+/", and both "Remove non-alphabet chars" and "Strict mode" checkboxes checked. The "Input" section on the right now contains the JWT string: `eyJ1c3VhcmIvIjoiriRG9uIFBlcGl0byBkZSBsb3MgcGFsb3RlcyIsInJvbmCI6ImIzTm9ybWFSliwiaWF0IjoxNjY3OTMzNTMzfQ.gfhw0dDxp6oixMLXXRP97W4TDTrv0y7B5YjD0U8ixrE`. The "Output" section at the bottom displays the decoded JSON: `{ "usuario": "Don Pepito de los palotes", "rol": "isNormal", "iat": 1667933533 }`. The status bar at the bottom shows "abc 99" and "1".

Un hacker está enviando a nuestro sistema el siguiente jwt:

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmVlIjoRG9uIFBlcGl0byBkZSBsb3MgcGFsb3RlcyIsInJvbCI6ImZlZWRtaW4iLCJpYXQiOiJlbnNjc5MzM1MzN9.krgBkzCBQ5WZ8JnZHuRvmnAZdg4ZMeRNV2CIAODIHRl

¿Qué está intentando realizar?

El hacker esta intentando modificar el contenido de JWT para cambiar el rol de usuario de **isNormal** a **isAdmin**



¿Qué ocurre si intentamos validarlo con pyjwt?

PyJWT indica que la firma no es válida.

<https://github.com/Andresricv/Main-Cripto/blob/main/Ejercicio%204%20HS256>.

Ejercicio 5.

El siguiente hash se corresponde con un SHA3 del texto “En KeepCoding aprendemos cómo protegernos con criptografía”.

bced1be95fbd85d2ffcce9c85434d79aa26f24ce82fbd4439517ea3f072d56fe

¿Qué tipo de SHA3 hemos generado?

El hash **bced1be95fbd85d2ffcce9c85434d79aa26f24ce82fbd4439517ea3f072d56fe** tiene 64 caracteres hexadecimales, lo que indica que tiene 256 bits. Por lo tanto, el tipo de SHA3 generado es **SHA3-256**.

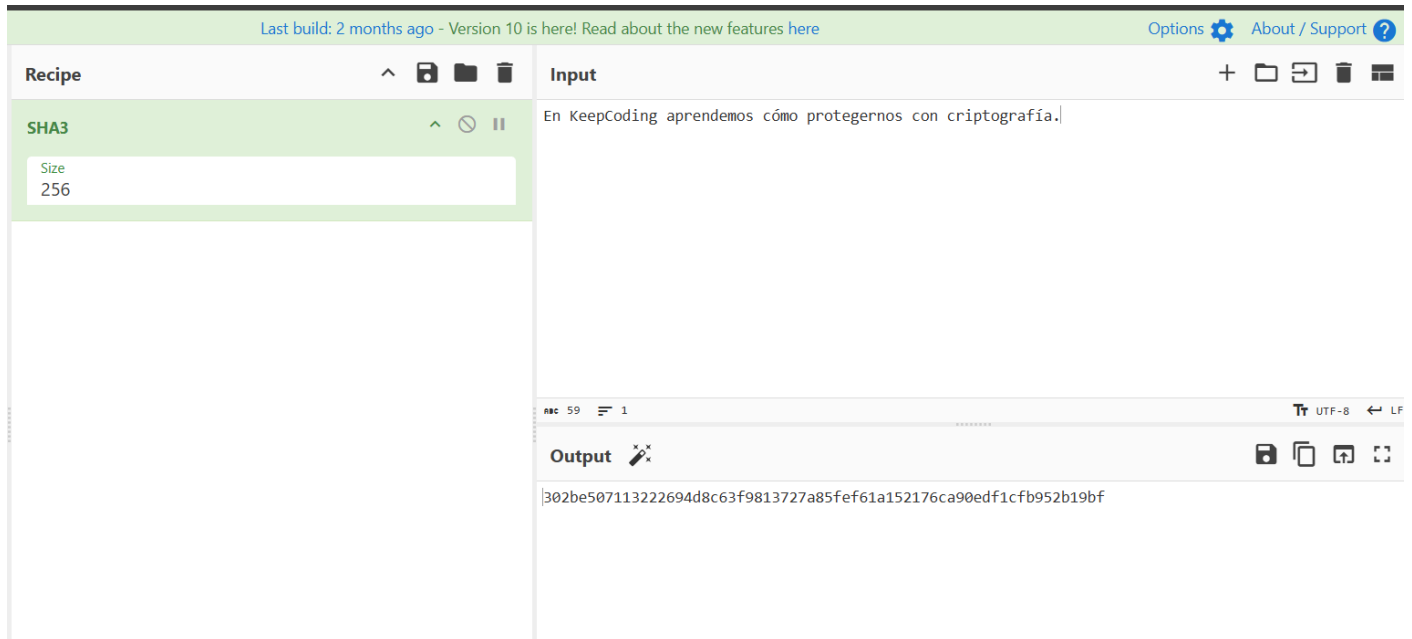
Y si hacemos un SHA2, y obtenemos el siguiente resultado:

4cec5a9f85dcc5c4c6ccb603d124cf1cdc6dfe836459551a1044f4f2908aa5d63739506f
6468833d77c07cfd69c488823b8d858283f1d05877120e8c5351c833

¿Qué hash hemos realizado?

contiene 128 caracteres hexadecimales. Esto indica que tiene 512 bits (128 × 4 bits). Por lo tanto, el hash realizado es **SHA-512**.

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



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.

```
1 import hmac
2 import hashlib
3
4 texto_objetivo = "Siempre existe más de una forma de hacerlo, y más de una solución válida."
5 clave_hex = "A212A51C997E14B4DF88055967641B8677CA31E049E672A4B86861AA4D5826EB"
6
7
8 clave_bytes = bytes.fromhex(clave_hex)
9
10 # Calcular el HMAC-SHA256
11 hmac_sha256 = hmac.new(clave_bytes, texto_objetivo.encode(), hashlib.sha256).hexdigest()
12
13
14 print(f"HMAC-SHA256: {hmac_sha256}")
```

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?

Vulnerabilidades criptográficas: SHA-1 tiene debilidades ante ataques de colisión, lo que significa que es posible generar dos entradas diferentes que producen el mismo hash. En 2017, se demostró un ataque práctico de colisión.

Velocidad: SHA-1 es muy rápido de calcular, lo que lo hace susceptible a ataques de fuerza bruta o ataques de diccionario mediante GPUs, FPGAs, o ASICs.

Obsolescencia: Muchas organizaciones y estándares de seguridad (como NIST) desaconsejan su uso desde hace años.

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?

Para fortalecerlo, es común aplicar un enfoque conocido como **key stretching** y usar un algoritmo diseñado específicamente para almacenar contraseñas. Algunas formas de fortalecer SHA-256:

-Salting: Añadir un valor aleatorio único (salt) a cada contraseña antes de calcular el hash. Esto previene ataques de diccionario y de tablas arcoíris.

-Iteración: Recalcular el hash repetidamente, lo que incrementa el costo computacional de los ataques. Un ejemplo sería realizar 100,000 iteraciones de SHA-256.

Parece que el responsable se ha quedado conforme, tras mejorar la propuesta del SHA-256, no obstante, hay margen de mejora. ¿Qué propondrías?

Usar un algoritmo moderno como Argon2.

Ejercicio 8.

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?

AES-256-GCM

Ejercicio 9.

Se requiere calcular el KCV de las siguiente clave AES:

A2CFF885901A5449E9C448BA5B948A8C4EE377152B3F1ACFA0148FB3A426DB72

Para lo cual, vamos a requerir el KCV(SHA-256) así como el KCV(AES). El KCV(SHA-256) se corresponderá con los 3 primeros bytes del SHA-256. Mientras que el KCV(AES) se corresponderá con cifrar un texto del tamaño del bloque AES (16 bytes) compuesto con ceros binarios (00), así como un iv igualmente compuesto de ceros binarios. Obviamente, la clave usada será la que queremos obtener su valor de control.

```
1 import hashlib
2 from base64 import b64encode, b64decode
3 from Crypto.Cipher import AES
4 from Crypto.Util.Padding import pad, unpad
Source Control (Ctrl+Shift+G)
5
6 #Cifrar una cadena de 16 bytes de ceros binarios.
7 textoPlano_bytes = bytes.fromhex('00000000000000000000000000000000')
8
9 clave = bytes.fromhex('A2CFF885901A5449E9C448BA5B948A8C4EE377152B3F1ACFA0148FB3A426DB72')
10 iv_bytes = bytes.fromhex('00000000000000000000000000000000')
11 cipher = AES.new(clave, AES.MODE_CBC, iv_bytes)
12 texto_cifrado_bytes = cipher.encrypt(pad(textoPlano_bytes, AES.block_size, style='pkcs7'))
13 print("KCV AES:", texto_cifrado_bytes.hex()[0:6])
14
15
16 m = hashlib.sha256()
17 m.update(bytes.fromhex("A2CFF885901A5449E9C448BA5B948A8C4EE377152B3F1ACFA0148FB3A426DB72"))
18 print("KCV SHA256: " + m.digest().hex()[0:6])
19
20
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

KCV AES: 5244db
KCV SHA256: db7df2

[Done] exited with code=0 in 0.097 seconds

[Running] python -u "/tmp/tempCodeRunnerFile.python"

KCV AES: 5244db
KCV SHA256: db7df2

[Done] exited with code=0 in 0.12 seconds

Ejercicio 10.

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, agosto, si no hay sorpresas. 11. Nuestra compañía ti

No he logrado dar con la solución de este ejercicio, tras varias horas de intento y haciéndolo a la vez que seguía el pdf de GPG y la clase en vivo, no he sido capaz, ya que me generaba distintos errores que no me dejaban avanzar en este ejercicio.

Ejercicio 11.

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:

```
b72e6fd48155f565dd2684df3ffa8746d649b11f0ed4637fc4c99d18283b32e1709b30c
96b4a8a20d5dbc639e9d83a53681e6d96f76a0e4c279f0dffa76a329d04e3d3d4ad629
793eb00cc76d10fc00475eb76bfbcb1273303882609957c4c0ae2c4f5ba670a4126f2f14
a9f4b6f41aa2edba01b4bd586624659fca82f5b4970186502de8624071be78ccef573d
896b8eac86f5d43ca7b10b59be4acf8f8e0498a455da04f67d3f98b4cd907f27639f4b1
df3c50e05d5bf63768088226e2a9177485c54f72407fdf358fe64479677d8296ad38c6f
177ea7cb74927651cf24b01dee27895d4f05fb5c161957845cd1b5848ed64ed3b0372
2b21a526a6e447cb8ee
```

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

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

Los valores son distintos porque el RSA-OAEP cifra con valores aleatorios añadiendo padding para generar una clave distinta aunque sea el mismo texto.

Ejercicio 12.

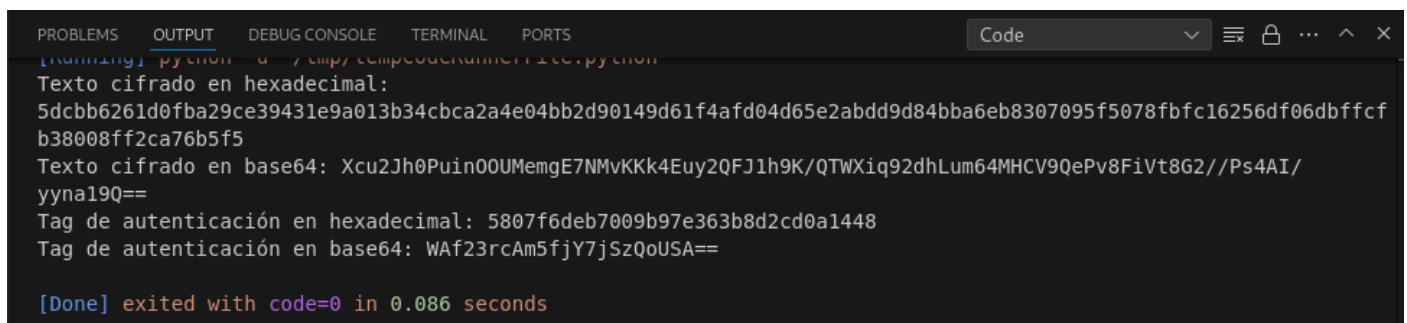
Nos debemos comunicar con una empresa, para lo cual, hemos decidido usar un algoritmo como el AES/GCM en la comunicación. Nuestro sistema, usa los siguientes datos en cada comunicación con el tercero:

Key: E2CFF885901B3449E9C448BA5B948A8C4EE322152B3F1ACFA0148FB3A426DB74

Nonce: 9Yccn/f5nJJhAt2S

¿Qué estamos haciendo mal? Que estamos usando el mismo nonce y este nunca puede ser fijo, debe ser aleatorio.

Cifra el siguiente texto: He descubierto el error y no volveré a hacerlo mal Usando para ello, la clave, y el nonce indicados. El texto cifrado presentalo en hexadecimal y en base64.



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  Code
[main@mg] python3 a.py /tmp/compendium/11c.py python
Texto cifrado en hexadecimal:
5dcbb6261d0fba29ce39431e9a013b34cbca2a4e04bb2d90149d61f4afd04d65e2abdd9d84bba6eb8307095f5078fbfc16256df06dbffcf
b38008ff2ca76b5f5
Texto cifrado en base64: Xcu2Jh0Puin00UMemgE7NMvKKk4Euy2QFJ1h9K/QTWxiq92dhLum64MHCv9QePv8FiVt8G2//Ps4AI/
yyna19Q==
Tag de autenticación en hexadecimal: 5807f6deb7009b97e363b8d2cd0a1448
Tag de autenticación en base64: WAF23rcAm5fjY7jSzQoUSA==

[Done] exited with code=0 in 0.086 seconds
```

Ejercicio 13.

Se desea calcular una firma con el algoritmo PKCS#1 v1.5 usando las claves contenidas en los ficheros clave-rsa-oaep-priv y clave-rsa-oaep-publ.pem del mensaje siguiente:

El equipo está preparado para seguir con el proceso, necesitaremos más recursos.

¿Cuál es el valor de la firma en hexadecimal?

Calcula la firma (en hexadecimal) con la curva elíptica ed25519, usando las claves ed25519-priv y ed25519-publ.

Ejercicio 14.

Necesitamos generar una nueva clave AES, usando para ello una HKDF (HMAC-based Extractand-Expand key derivation function) con un hash SHA-512. La clave maestra requerida se encuentra en el keystore con la etiqueta "cifrado-sim-aes-256". La clave obtenida dependerá de un identificador de dispositivo, en este caso tendrá el valor en hexadecimal:

e43bb4067cbcfab3bec54437b84bef4623e345682d89de9948fbb0afedc461a3

¿Qué clave se ha obtenido?

f6fbc6204bd24b43c42fe1be7d970eeecbcee87481711a64433ea1b7ef655ffa

Ejercicio 15.

Nos envían un bloque TR31:

D0144D0AB00S000042766B9265B2DF93AE6E29B58135B77A2F616C8D515ACDB
E6A5626F79FA7B4071E9EE1423C6D7970FA2B965D18B23922B5B2E5657495E0
3CD857FD37018E111B Donde la clave de transporte para desenvolver (unwrap)

el bloque es: A1A1010101010101010101010101010102

¿Con qué algoritmo se ha protegido el bloque de clave? AES

¿Para qué algoritmo se ha definido la clave? AES

¿Para qué modo de uso se ha generado? Cifrado y descifrado

¿Es exportable? Si, bajo una untrusted key

¿Para qué se puede usar la clave? Cifrar datos

Cifrar/descifrar claves de datos usando algoritmos como 3DES o AES.

¿Qué valor tiene la clave? C1c1c1c1c1c1c1c1c1c1c1c1c1c1c1