**Alumno: José Benito Ibarria Topete**

**PRÁCTICA: CRIPTOGRAFÍA**

**Objetivo:**

Contestar un conjunto de preguntas / ejercicios relacionados con la materia aprendida en el curso demostrando la adquisición de conocimientos relacionados con la criptografía.

**Detalles:**

En esta práctica el alumno aplicará las técnicas y utilizará las diferentes herramientas vistas durante el módulo.

Cualquier password que sea necesaria tendrá un valor 123456.

**Evaluación**

Es obligatorio la entrega de un informe para considerar como APTA la práctica. Este informe ha de contener:

* Los enunciados seguido de las respuestas justificadas y evidenciadas.
* En el caso de que se hayan usado comandos / herramientas también se deben nombrar y explicar los pasos realizados.

El código escrito para la resolución de los problemas se entrega en archivos separados junto al informe.

Se va a valorar el proceso de razonamiento aunque no se llegue a resolver completamente los problemas. Si el código no funciona, pero se explica detalladamente la intención se valorará positivamente.

El objetivo principal de este módulo es adquirir conocimientos de criptografía y por ello se considera fundamental usar cualquier herramienta que pueda ayudar a su resolución, demostrando que no sólo se obtiene el dato sino que se tiene un conocimiento profundo del mismo. Si durante la misma no se indica claramente la necesidad de resolverlo usando programación, el alumno será libre de usar cualquier herramienta, siempre y cuando aporte las evidencias oportunas.

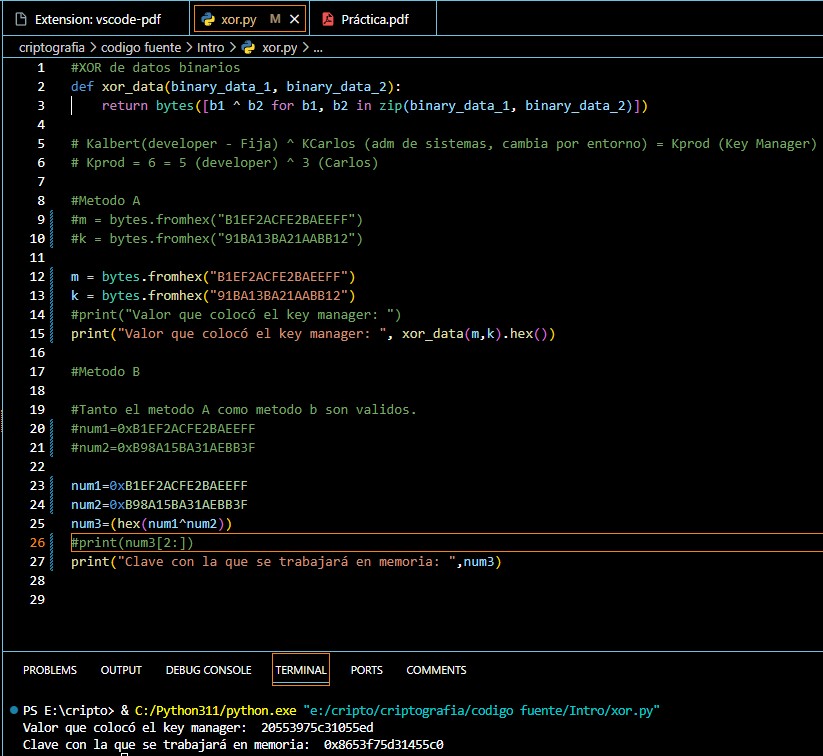
**Ejercicios:**

**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?

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?

***R.-*** En este caso nos apoyaremos con el programa xor.py para realizar ambas operaciones XOR al mismo tiempo:



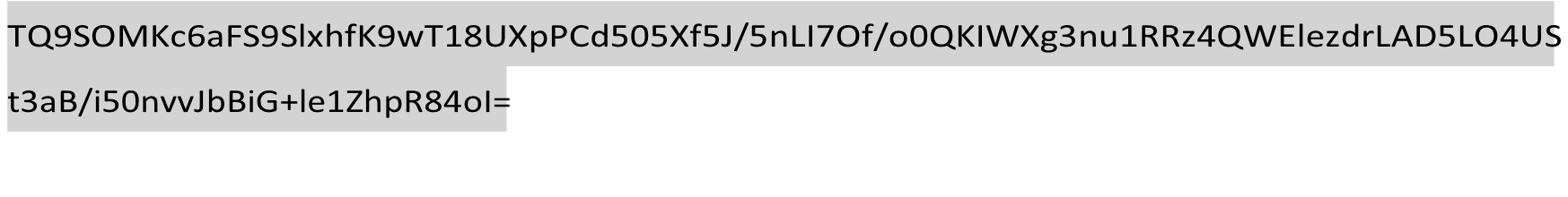
Los resultados obtenidos mediante el programa xor.py son los siguientes:

1. El key manager ha puesto el siguiente valor en properties para forzar la clave final: ***20553975c31055ed***

1. La clave en producción, con la que se trabajará en memoria es:

***8653f75d31455c0***

**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:



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

**R.-** Obtenemos el siguiente mensaje: “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?

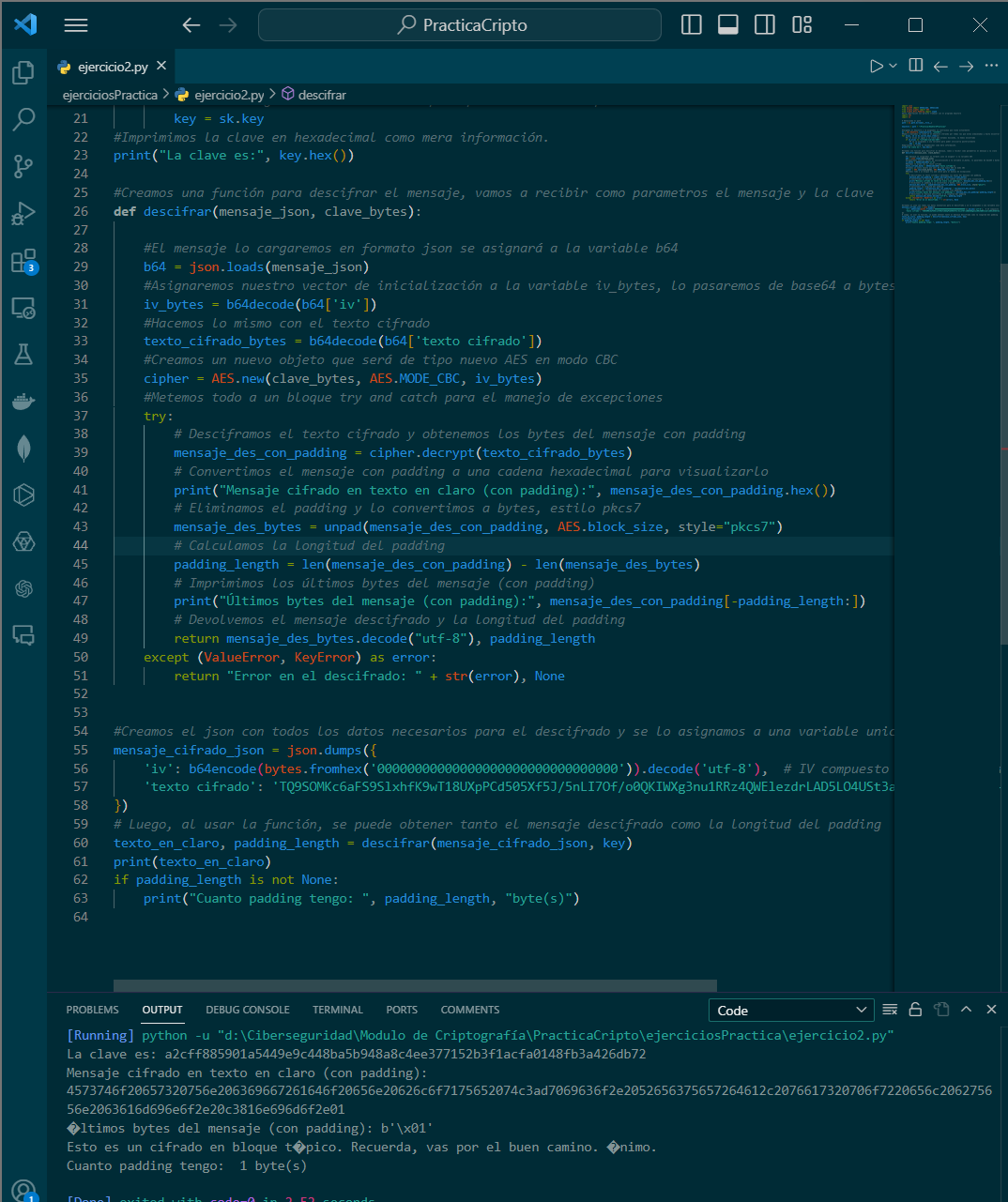
**R.-** Aunque lo lógico sería que tuviéramos un error ya que el cifrado se ha realizado bajo el estilo “PKCS7”, en realidad no tenemos ningún problema ya que, en esencia son de tipo iguales puesto que al final se añade el número del padding adicionado en el formato 01 en el tipo x923, dado nuestro ejemplo de práctica, pasaría lo mismo si usáramos el tipo de padding ISO 101 26. No debemos de olvidar que solo en el caso de nuestro ejemplo de practica funcionaría(cuando termina en 01), en el resto de los casos mandaría un error y se debería de colocar forzosamente el mismo estilo de padding en el cifrado y en el descifrado.

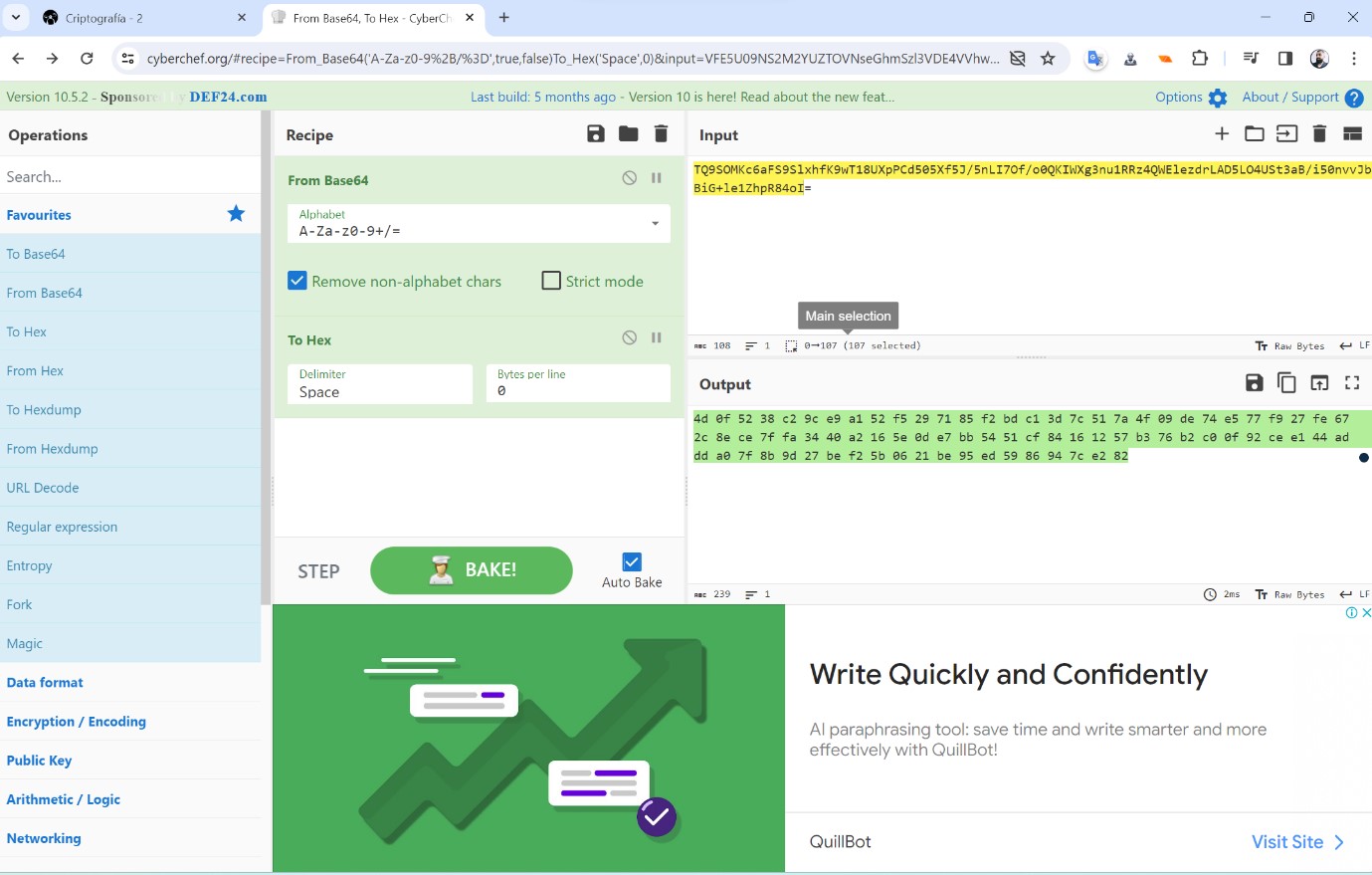
¿Cuánto padding se ha añadido en el cifrado?

**R.-** Se ha añadido un padding de 1 byte, esto porque hemos añadido al flujo de ejecución del programa la impresión del mensaje cifrado en texto en claro antes de añadirle el padding, mostrándome en pantalla “x01” que fundamentalmente en el estilo de padding pkcs significa 1 byte de padding, ademas hemos realizado una resta del mensaje con padding – los bytes que tiene el mensaje dando un total de 1 byte y sumado a lo anterior, hacemos en cybercheff la conversión de base64 del texto cifrado a hexadecimal y nos dá un total de 80 bytes, eso nos sirve porque entonces podemos determinar que el tamaño del bloque es 16 bytes (El algoritmo AES utiliza tamaños de bloque de 16 bytes, por ejemplo 3DES utiliza un tamaño de bloque de 8 bytes) y para confirmar, al seleccionar el bloque cifrado en hexadecimal solo se seleccionan los caracteres del texto cifrado desde la “T” y hasta la “l” sobrando el signo “=”, siendo este el padding de 1 byte.

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



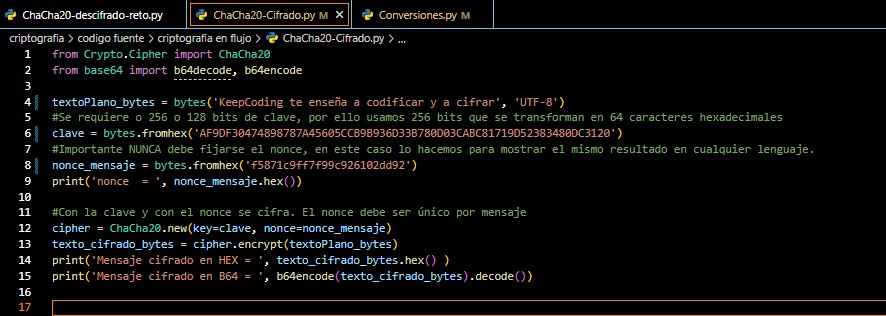




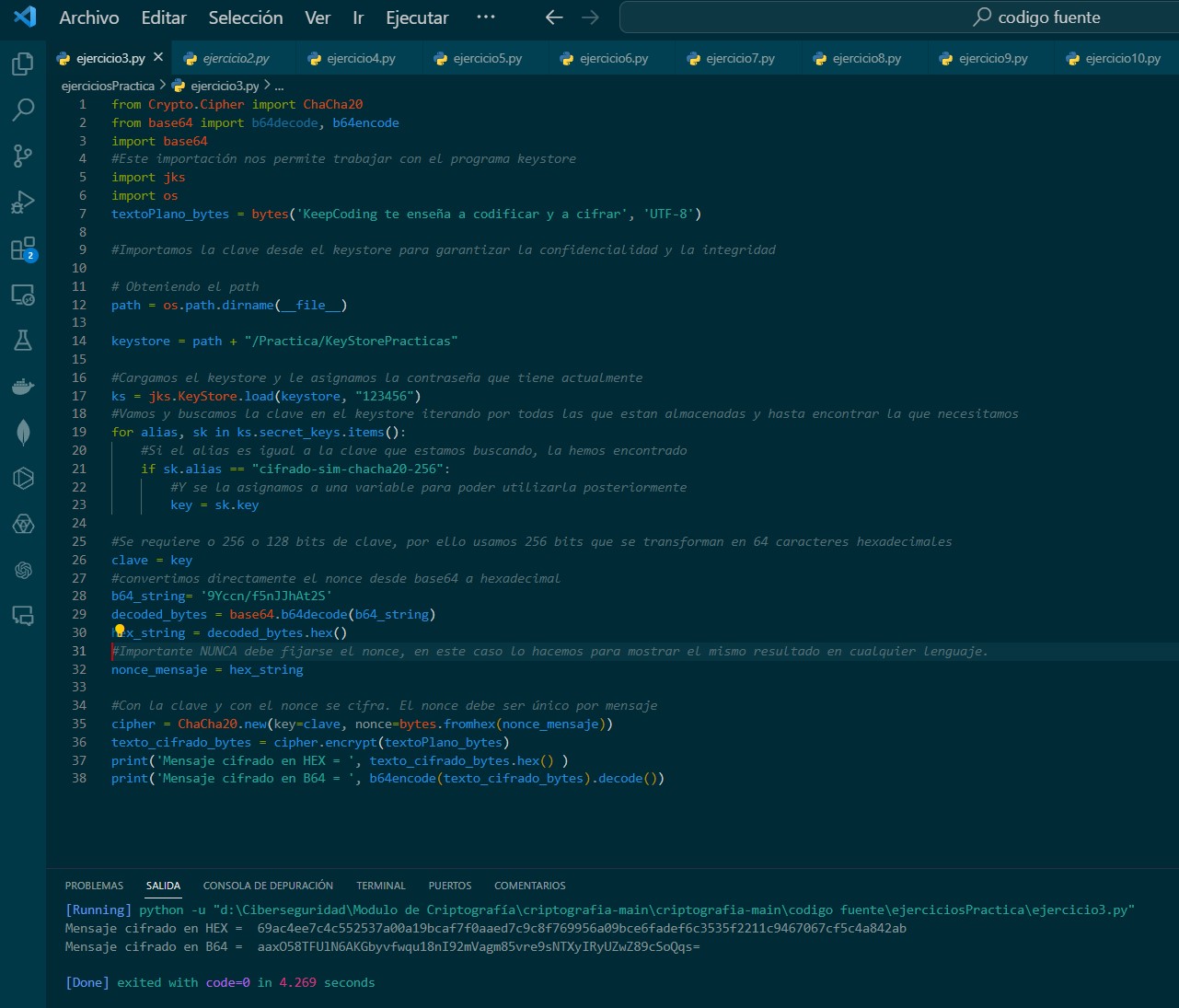
**3.** Se requiere cifrar el texto “KeepCoding te enseña a codificar y a cifrar”. La clave para ello, tiene la etiqueta en el Keystore “cifrado-sim-chacha-256”. El nonce “9Yccn/f5nJJhAt2S”. El algoritmo que se debe usar es un Chacha20.

¿Cómo podríamos mejorar de forma sencilla el sistema, de tal forma, que no sólo garanticemos la confidencialidad sino, además, la integridad de este?

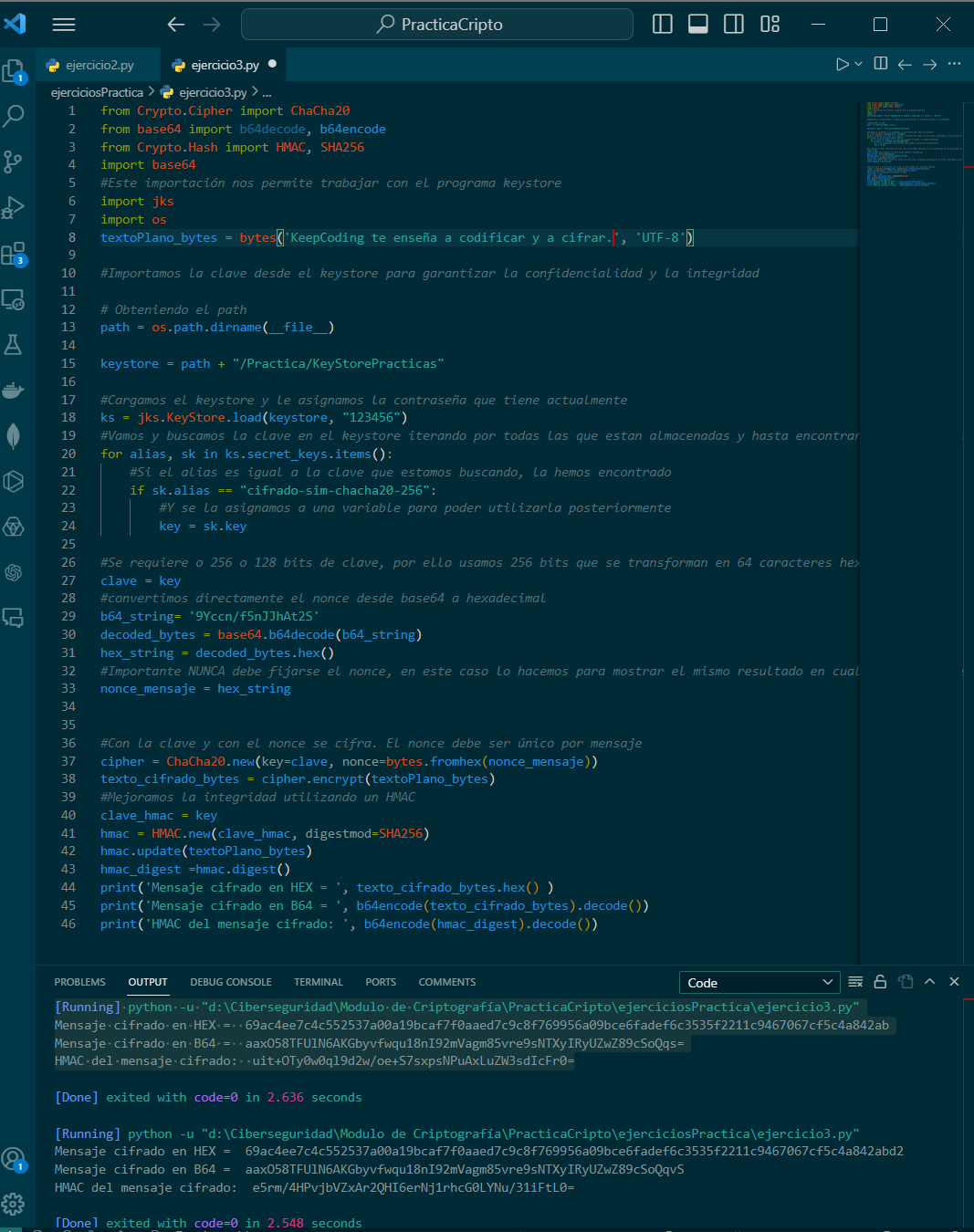
**R. –** Considero que podríamos mejorar de forma sencilla el sistema haciéndolo más robusto extrayendo la clave directamente desde el Keystore y que no la coloquemos de forma manual en el código fuente ya que esta podría ser vista de forma directa por un actor malicioso, por lo que considero que esta clave debería de estar oculta bajo otra aplicación y esa aplicación con una contraseña que también sea robusta, así mismo, podríamos añadir al bloque de código un nuevo bloque que realice la conversión del “nounce” desde base64 a hexadecimal y pasar ese valor de forma directa al cifrado en la variable “cipher”, sumado a todo lo anterior podríamos también calcular un HMAC para el mensaje cifrado y así poder proveer de una mayor integridad puesto que si alguien nos cambie cualquier cosa en el mensaje se calcule un HMAC distinto confirmando así una posible corrupción o ataque a la integridad de nuestro mensaje.



c) Se requiere obtener el dato cifrado, demuestra, tu propuesta por código, así como añadir los datos necesarios para evaluar tu propuesta de mejora.



Mejora con el cálculo del HMAC



Un punto al final del mensaje que cambié y tenemos un HMAC distinto, vale, con esta mejora podemos garantizar la integridad al mismo tiempo.

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

|  |  |
| --- | --- |
| eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmlvIjoiRG9uIFBlcGl0byBkZSB | |
| sb3MgcGFsb3RlcyIsInJvbCI6ImlzTm9ybWFsIiwiaWF0IjoxNjY3OTMzNTMzfQ.gfhw0 | |
| dDxp6oixMLXXRP97W4TDTrv0y7B5YjD0U8ixrE |  |

¿Qué algoritmo de firma hemos realizado?

{"typ":"JWT","alg":"HS256"}

***R.-*** De acuerdo a lo extraído desde ciberchef del header desde un “Frombase64” obtenemos que el algoritmo de firma usado es un HMAC + SHA256

¿Cuál es el body del jwt?

***R.-*** El body del JWT es el siguiente:

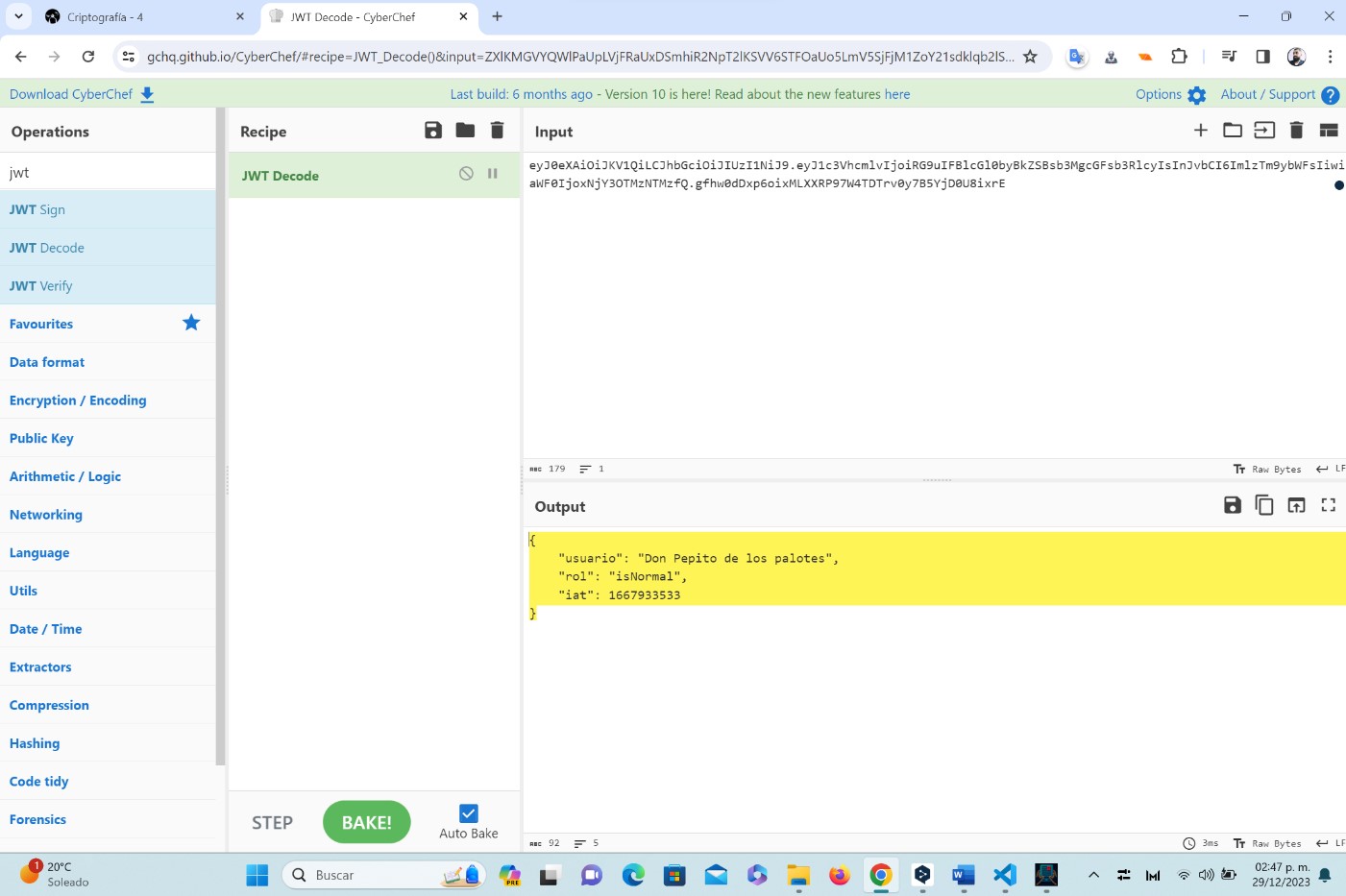
{

"usuario": "Don Pepito de los palotes",

"rol": "isNormal",

"iat": 1667933533

}



Un hacker está enviando a nuestro sistema el siguiente jwt:

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmlvIjoiRG9uIFBlcGl0byBk

ZSBsb3MgcGFsb3RlcyIsInJvbCI6ImlzQWRtaW4iLCJpYXQiOjE2Njc5MzM1MzN9

.krgBkzCBQ5WZ8JnZHuRvmnAZdg4ZMeRNv2CIAODlHRI

¿Qué está intentando realizar?

***R.-*** Está intentando básicamente el hacker autentificarse como administrador para poder quizás ejecutar un ataque de escalada de privilegios porque de acuerdo a la captura de pantalla obtenida desde ciberchef el payload es el siguiente:

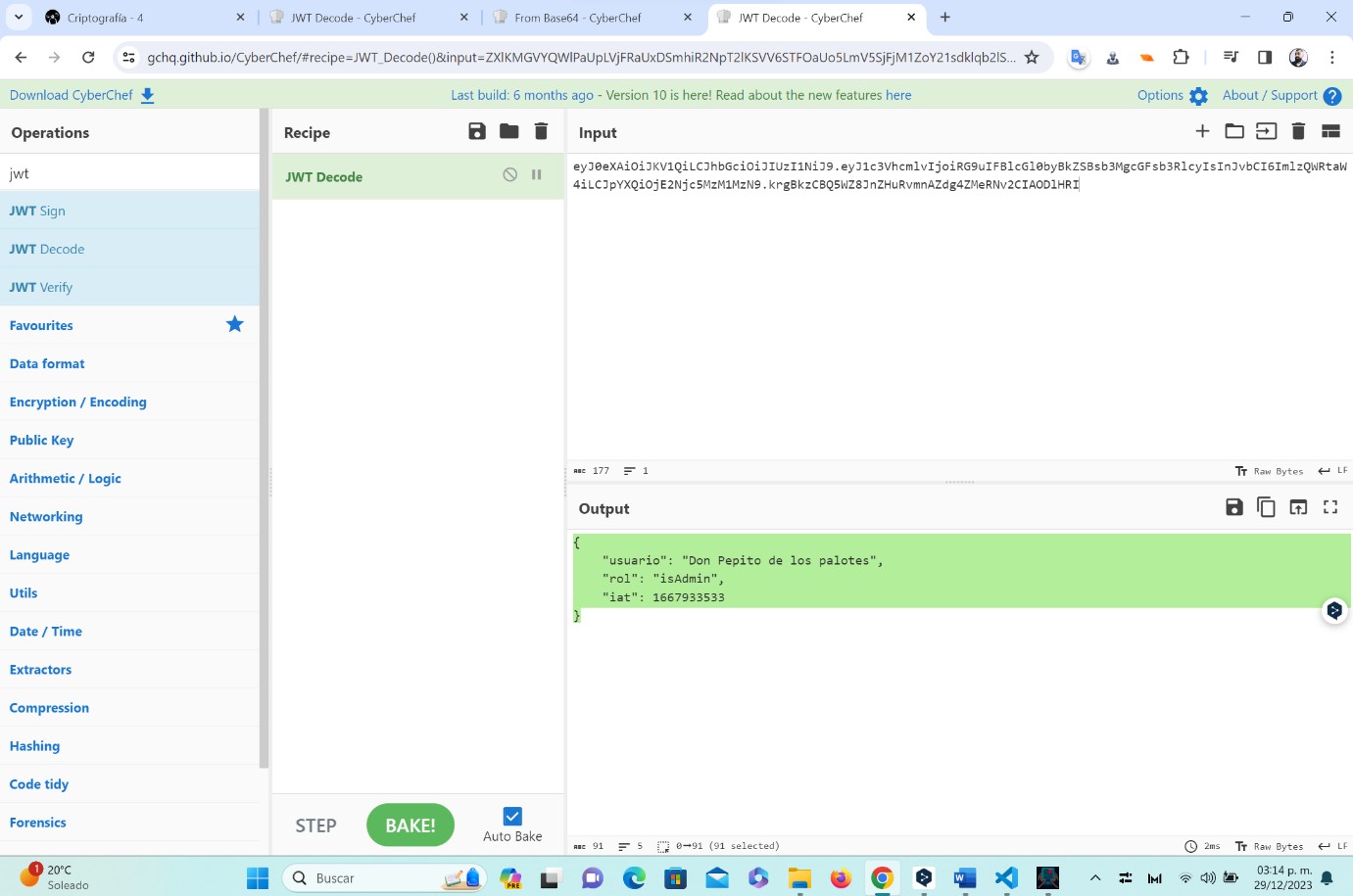
{

"usuario": "Don Pepito de los palotes",

"rol": "isAdmin",

"iat": 1667933533

}



¿Qué ocurre si intentamos validarlo con pyjwt?

***R.-*** Va a enviar un error porque con las versiones actuales de hacen una validación con la clave que se ha dado, si el hacker no tiene la clave Con KeepCoding aprendemos” Entonces no se va a poder autenticar

**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?

***R.-*** Hemos generado un tipo de SHA3\_256 bits porque la cadena de hash lograda (bced1be95fbd85d2ffcce9c85434d79aa26f24ce82fbd4439517ea3f072d56fe) tiene 64 caracteres que divididos entre 2 son 32 y estos 32 multiplicados por 8 nos da 256 bits

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

|  |  |
| --- | --- |
| 4cec5a9f85dcc5c4c6ccb603d124cf1cdc6dfe836459551a1044f4f2908aa5d63739506f | |
| 6468833d77c07cfd69c488823b8d858283f1d05877120e8c5351c833 |  |

¿Qué hash hemos realizado?

***R.-*** Hemos realizado un SHA-512 ya que el hash tiene 128 caracteres que divididos entre 2 da 64 y multiplicado por 8 encontramos un valor de 512 bits de salida.



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?

***R.-*** Destacaría el efecto difusión y el efecto avalancha atendiendo los resultados anteriores ya que el texto en este ejercicio lleva un carácter más en forma de un punto al final del texto utf8, esto lo hace que se comporte de manera muy distinta a los resultados obtenidos anteriormente, al menos con el SHA3 Keccak de 256 bits que hicimos al principio del ejercicio.

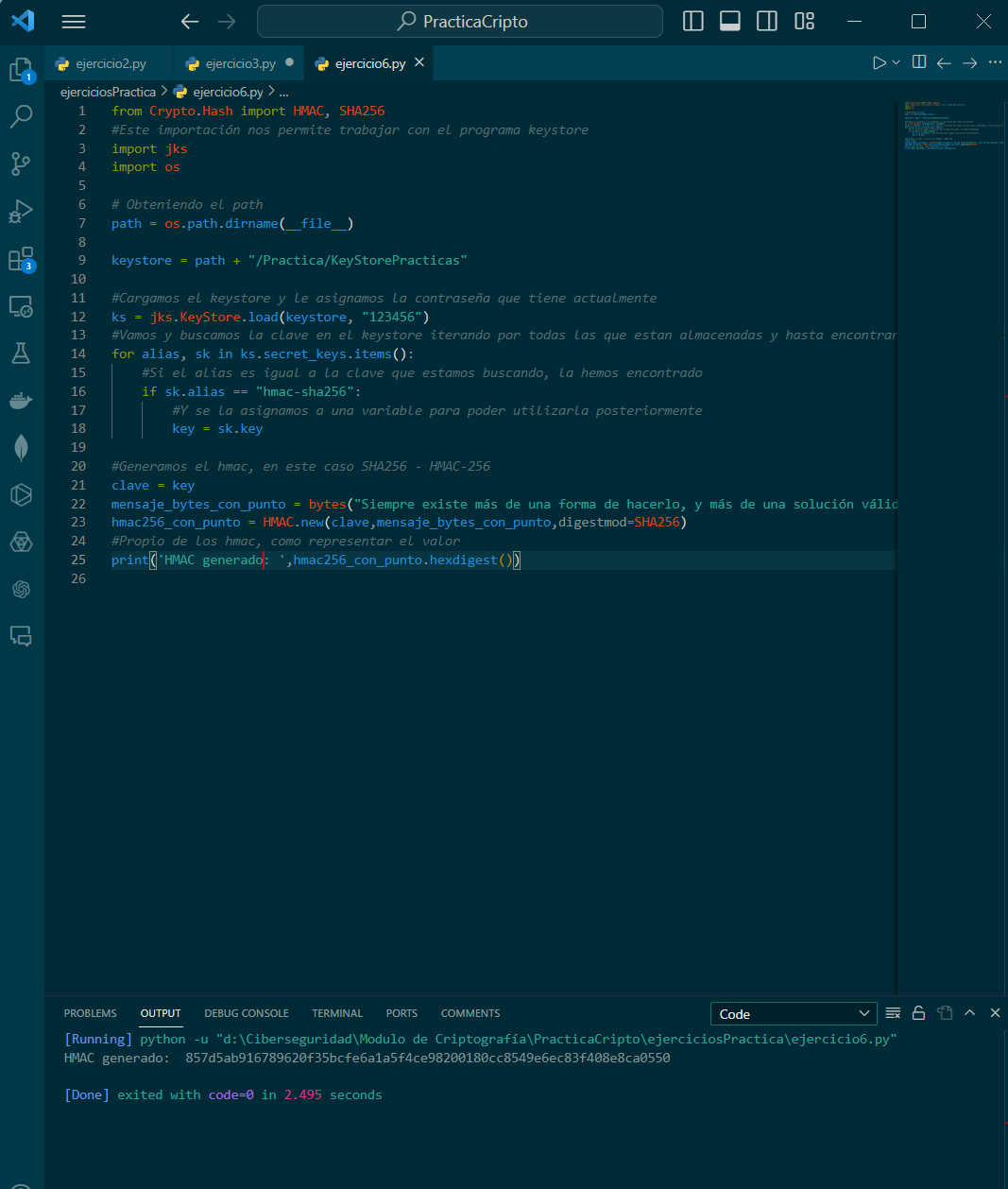
1. Calcula el hmac-256 (usando la clave contenida en el Keystore) del siguiente texto:

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

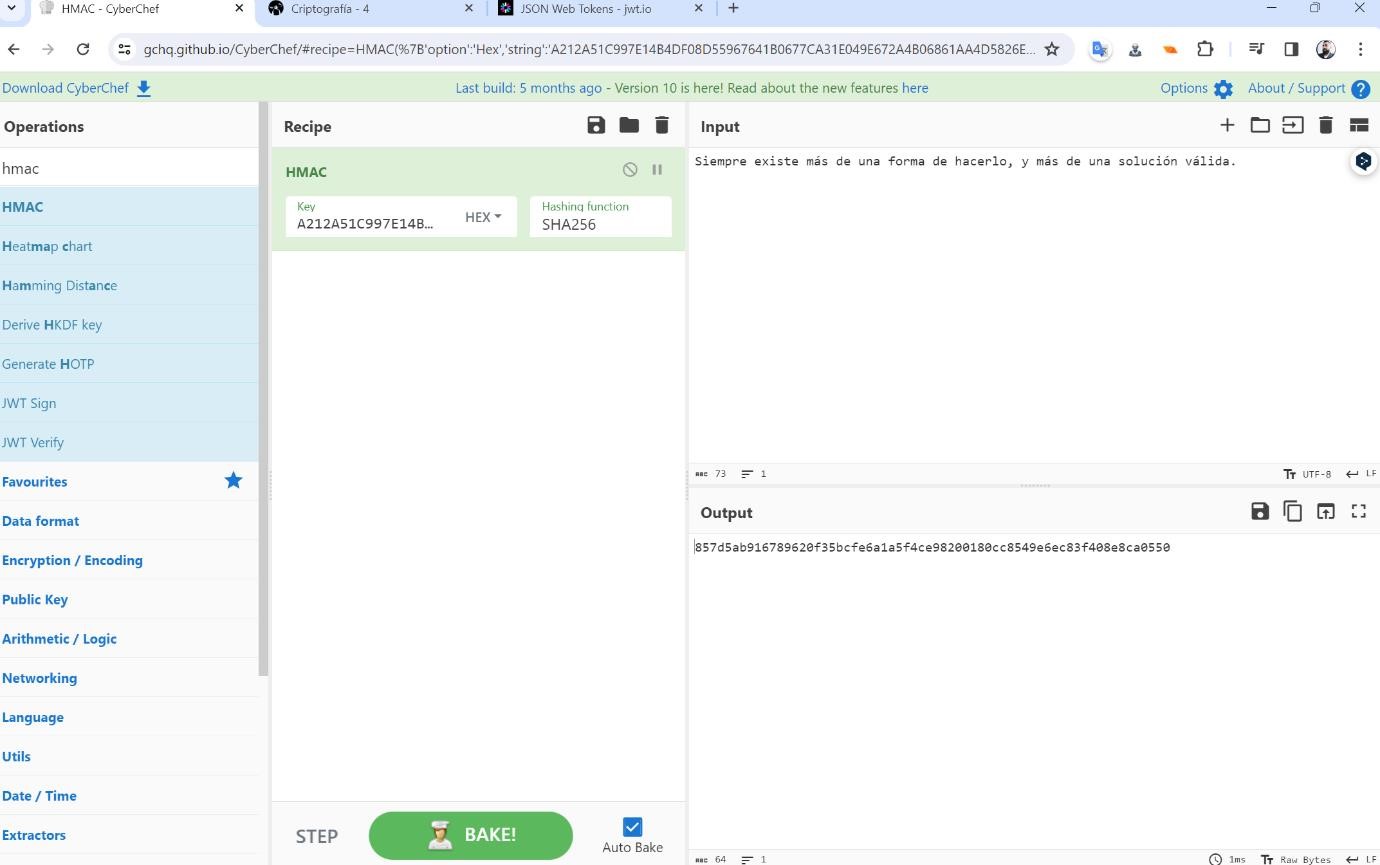
Se debe evidenciar la respuesta. Cuidado si se usan herramientas fuera de los lenguajes de programación, por las codificaciones es mejor trabajar en hexadecimal.

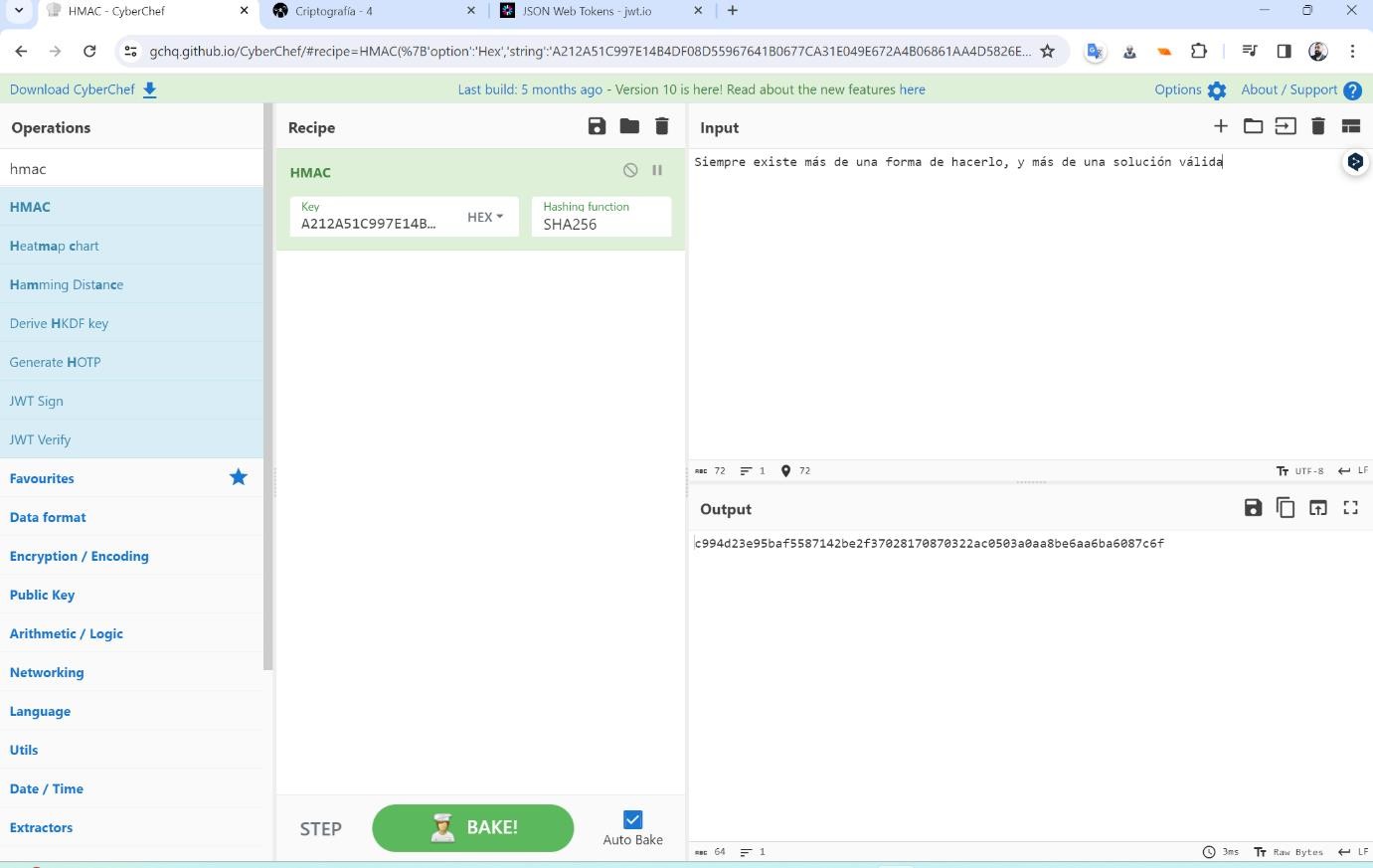
**R.-** En esencia tenemos el siguiente HMAC con el texto + punto final: HMAC

:857d5ab916789620f35bcfe6a1a5f4ce98200180cc8549e6ec83f408e8ca0550



Realizado con ciberchef:





1. 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?

***R.-*** Es una mala opción porque el algoritmo SHA-1 está deprecado a nivel de criptografía debido a que, hoy en día con el poder computacional actual podemos realizar el cálculo desde el valor del hash en hexadecimal a utf8, es decir, podemos ver el valor de forma reversible.

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?

***R.-*** Definitivamente se me ocurriría proponer agregar un valor aleatorio a los passwords de cada uno de los usuarios este valor aleatorio sería lo equivalente a agregar un valor de SALT + PEPPER a nuestro hash 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?

***R.-*** Como propuesta de mejora yo propondría añadir otro número aleatorio, único para la base de datos que contiene a todos los valores de los passwords. Dicho número sería, igual, un numero aleatorio pero en forma de “Pepper” que guardaríamos lejos de los hashes y passwords del usuario, y además, para mejorar la resistencia utilizaría Argon2id como Hash.

**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?

***R.-*** Considero como una excelente opción para la protección de contraseñas y los nombres de usuarios en la base de datos al algoritmo de hashing ARGON, fundamentalmente bajo el contexto de almacenar y verificar contraseñas de manera segura. Para el caso del cifrado de los campos como los números de las tarjetas propongo utilizar un algoritmo de cifrado simétrico como AES con GCM.

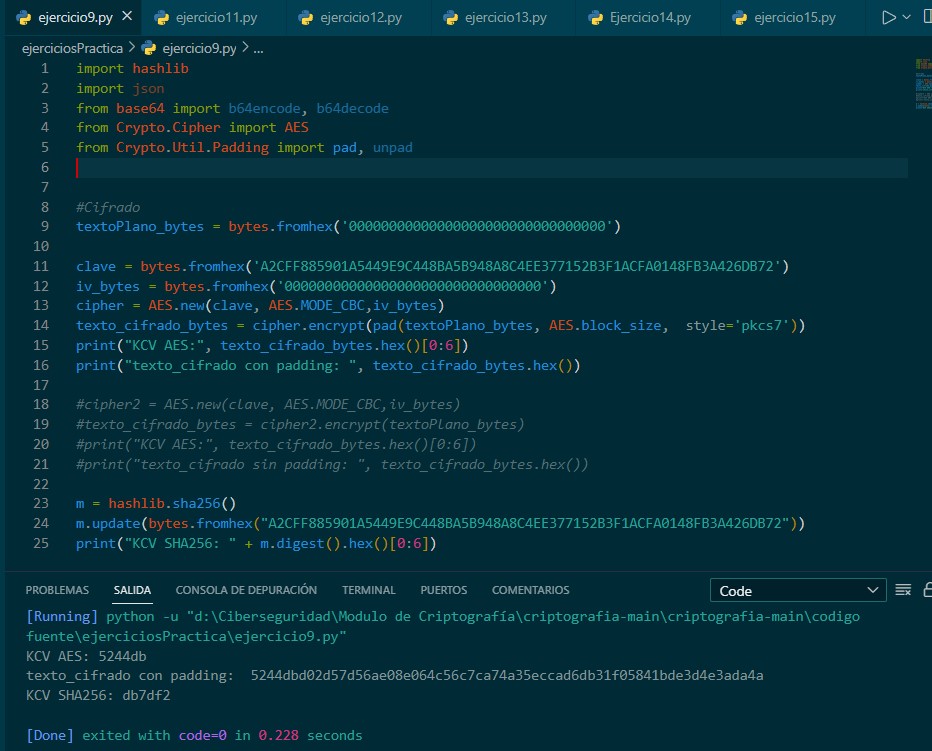
1. Se requiere calcular el KCV de las siguiente clave AES:

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.

***R.-*** El KCV(SHA256) es db7df2

***R.-*** El KCV(AES) es 5244db



1. 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.

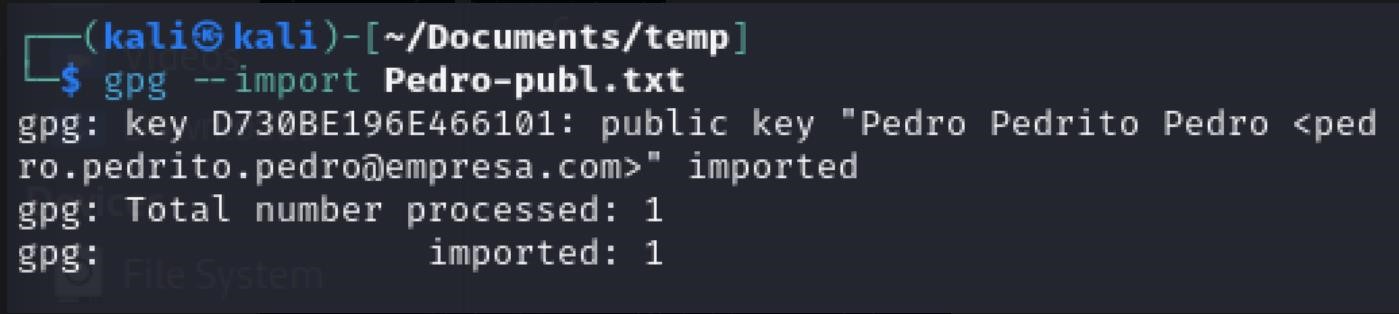
|  |  |
| --- | --- |
| Viendo su perfil en el mercado, hemos decidido ascenderle y mejorarle un 25% su | |
| salario. Saludos. |  |

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

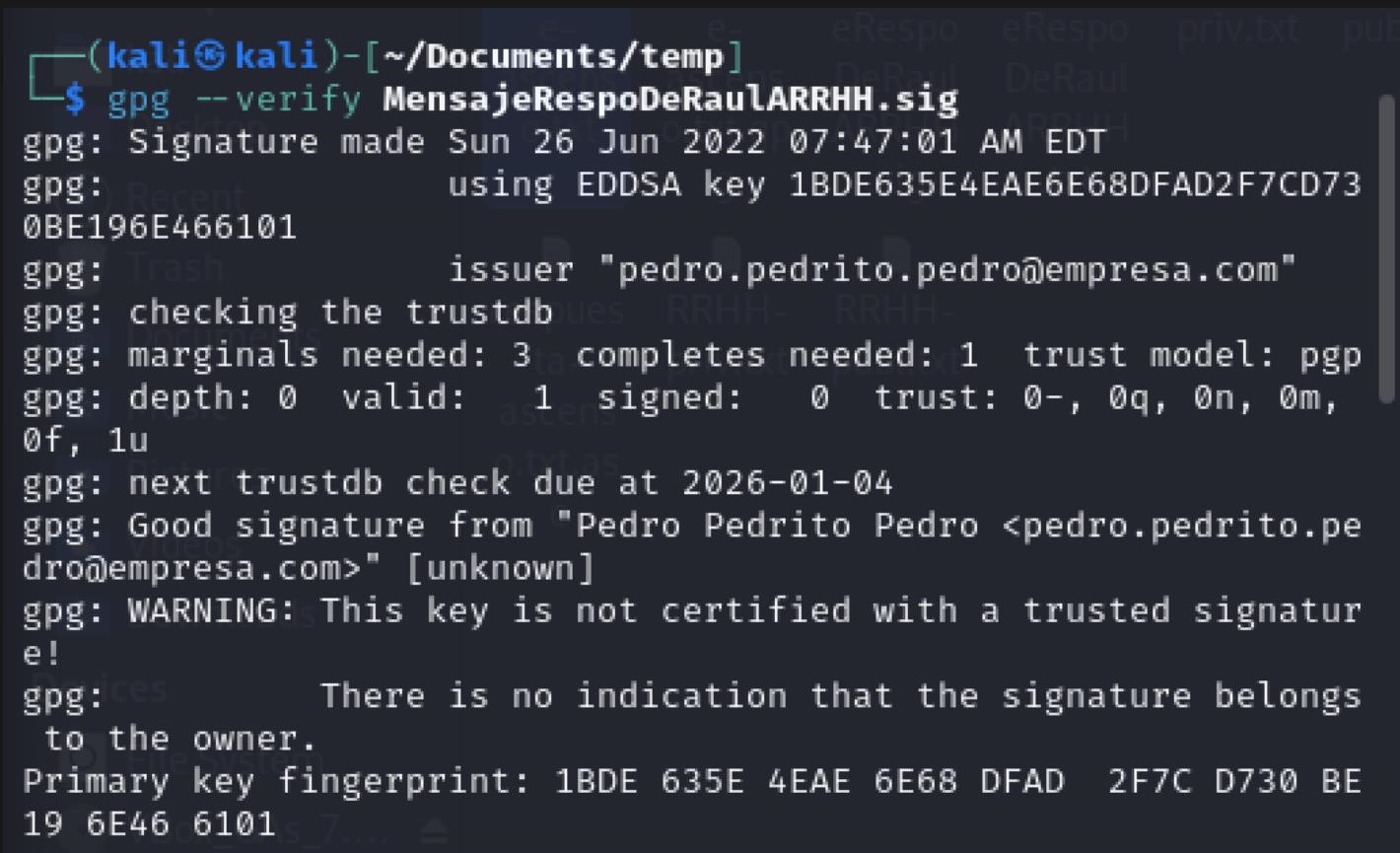
|  |  |
| --- | --- |
| Estamos todos de acuerdo, el ascenso será el mes que viene, agosto, si no hay | |
| sorpresas. |  |

***R.-*** Fundamentalmente hemos realizado el cifrado y la firma de los mensajes mediante gpg desde Kali Linux, la evidencia es la siguiente:

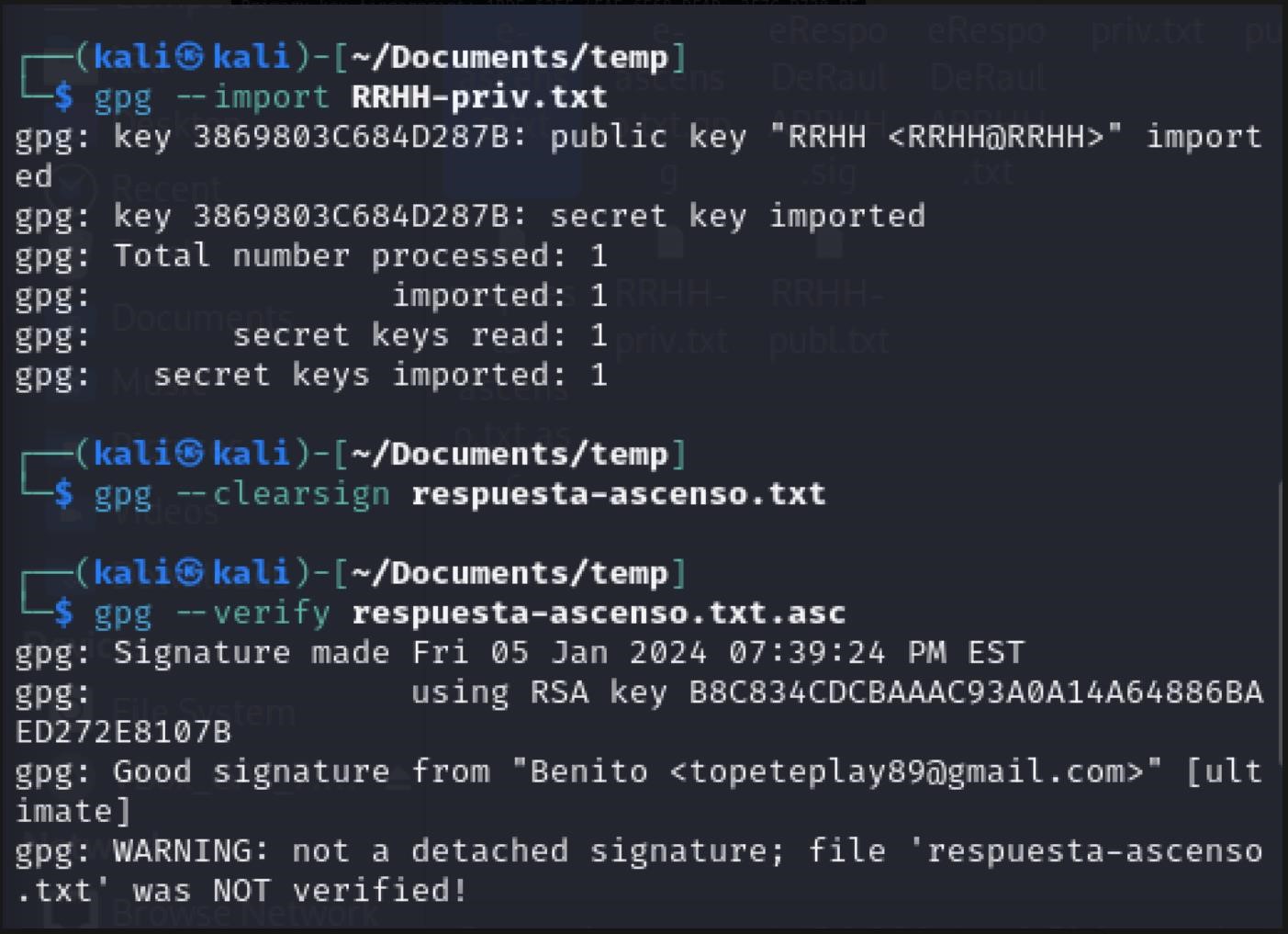
Importamos la clave pública de Pedro



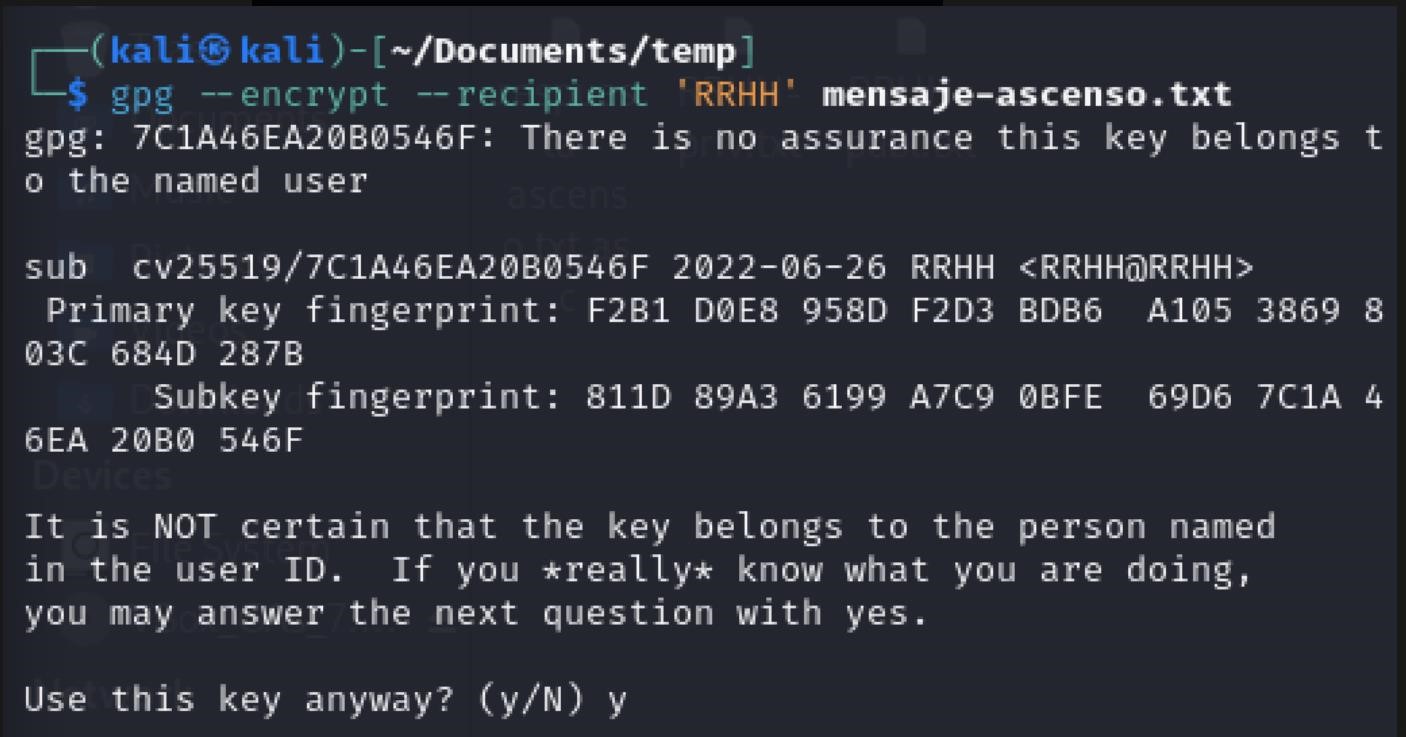
Verificamos que el mensaje haya sido efectivamente firmado por pedro



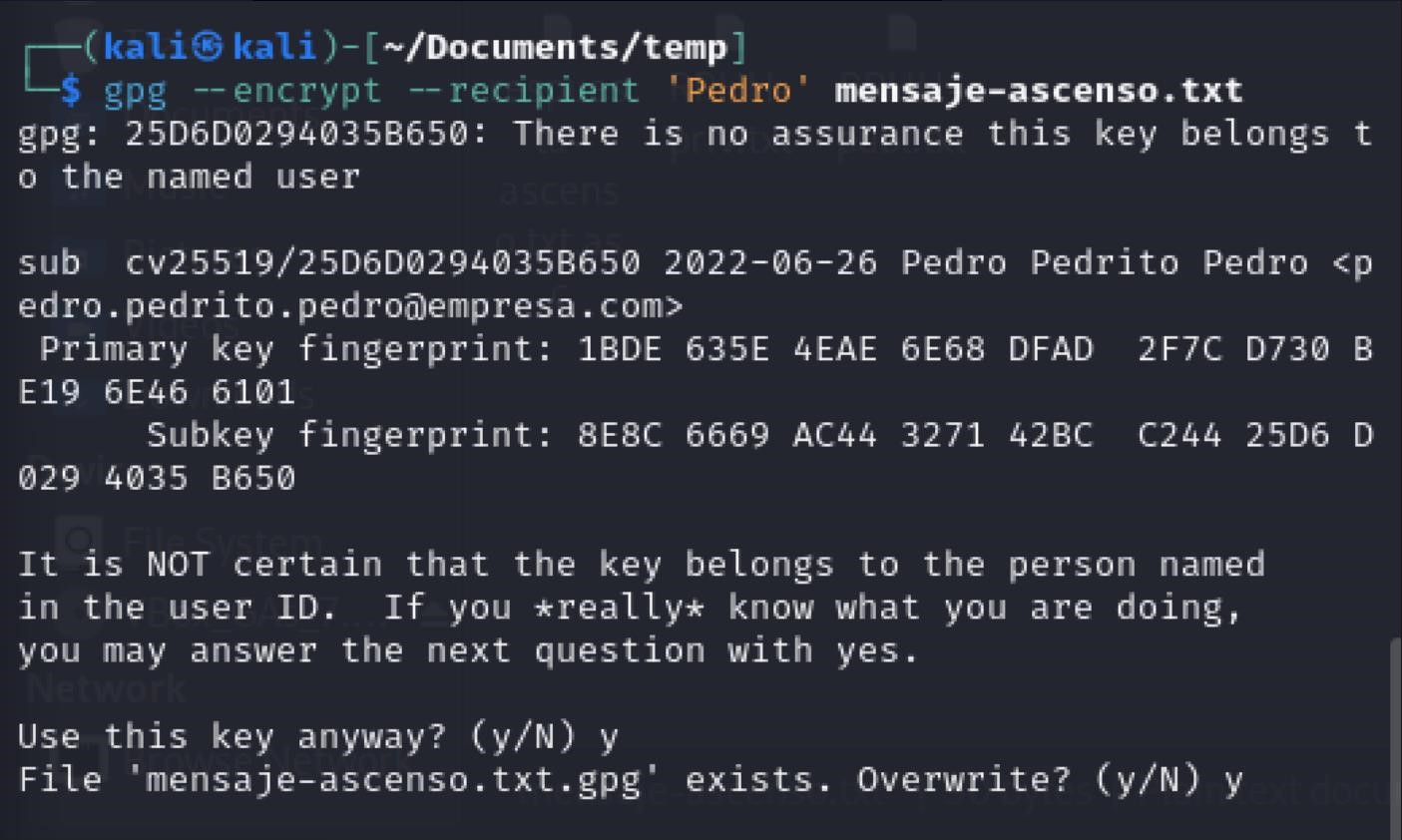
Importamos la clave privada de recursos humanos



Firmamos el mensaje de RRHH, verificamos la firma y confirmamos que efectivamente el mensaje viene desde recurso humanos



Ciframos el mensaje con la clave publica de Pedro



**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 793eb00cc76d10fc00475eb76bfbc1273303882609957c4c0ae2c4f5ba670a4126f2f14 a9f4b6f41aa2edba01b4bd586624659fca82f5b4970186502de8624071be78ccef573d 896b8eac86f5d43ca7b10b59be4acf8f8e0498a455da04f67d3f98b4cd907f27639f4b1 df3c50e05d5bf63768088226e2a9177485c54f72407fdf358fe64479677d8296ad38c6f



177ea7cb74927651cf24b01dee27895d4f05fb5c161957845cd1b5848ed64ed3b0372

2b21a526a6e447cb8ee

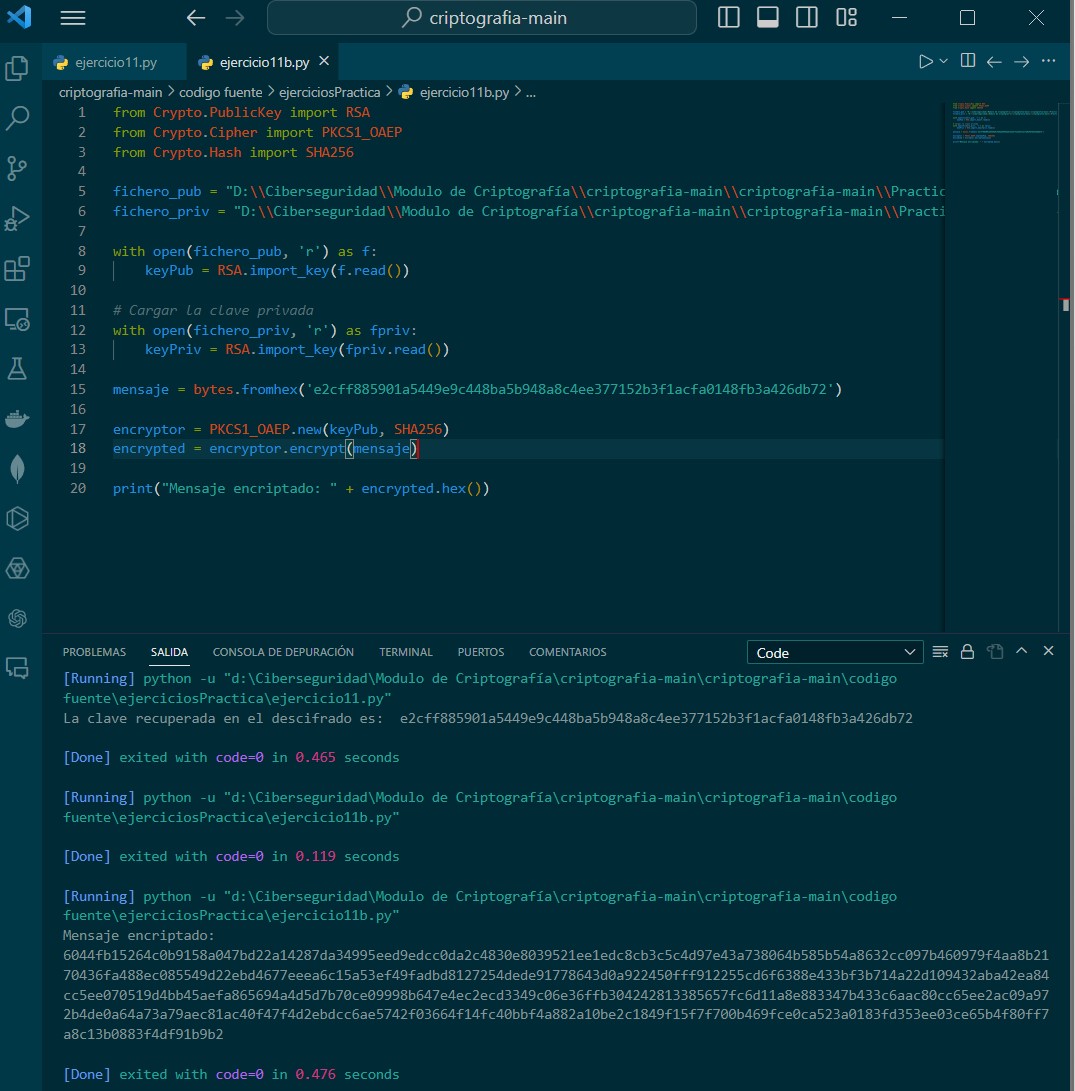
Las claves pública y privada las tenemos en los ficheros clave-rsa-oaep-publ.pem y claversaoaep-priv.pem.

***Primero desciframos bajo el algoritmo de criptografía asimétrica RSA-OAEP***

***Clave Recuperada:***

e2cff885901a5449e9c448ba5b948a8c4ee377152b3f1acfa0148fb3a426db72

***Segundo, ciframos bajo el algoritmo de criptografía asimétrica RSA-OAEP utilizando la clave obtenida en el descifrado anterior:***



***Obteniendo el siguiente mensaje en la primera ejecución:***

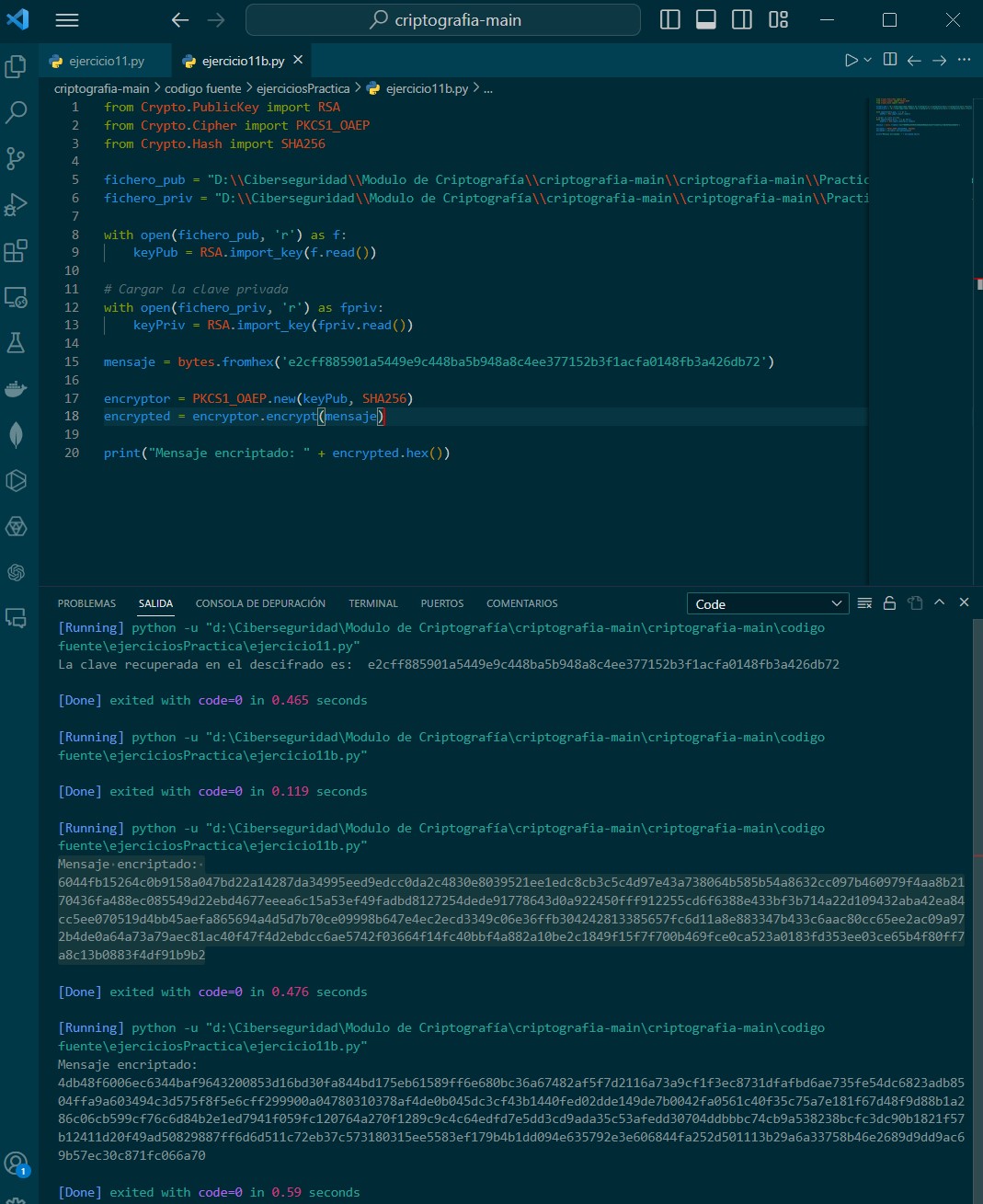
Mensaje encriptado:

6044fb15264c0b9158a047bd22a14287da34995eed9edcc0da2c4830e8039521ee1edc8cb3c5c4 d97e43a738064b585b54a8632cc097b460979f4aa8b2170436fa488ec085549d22ebd4677eeea6 c15a53ef49fadbd8127254dede91778643d0a922450fff912255cd6f6388e433bf3b714a22d109 432aba42ea84cc5ee070519d4bb45aefa865694a4d5d7b70ce09998b647e4ec2ecd3349c06e36f fb304242813385657fc6d11a8e883347b433c6aac80cc65ee2ac09a972b4de0a64a73a79aec81a c40f47f4d2ebdcc6ae5742f03664f14fc40bbf4a882a10be2c1849f15f7f700b469fce0ca523a0

183fd353ee03ce65b4f80ff7a8c13b0883f4df91b9b2

|  |
| --- |
|  |

***Tercero, lo volvemos a ejecutar utilizando las mismas claves (Tanto pública como privada y la obtenida en el descifrado)***



***Obteniendo el siguiente mensaje cifrado:***

4db48f6006ec6344baf9643200853d16bd30fa844bd175eb61589ff6e680bc36a67482af5f7d21

16a73a9cf1f3ec8731dfafbd6ae735fe54dc6823adb8504ffa9a603494c3d575f8f5e6cff29990 0a04780310378af4de0b045dc3cf43b1440fed02dde149de7b0042fa0561c40f35c75a7e181f67 d48f9d88b1a286c06cb599cf76c6d84b2e1ed7941f059fc120764a270f1289c9c4c64edfd7e5dd 3cd9ada35c53afedd30704ddbbbc74cb9a538238bcfc3dc90b1821f57b12411d20f49ad5082988 7ff6d6d511c72eb37c573180315ee5583ef179b4b1dd094e635792e3e606844fa252d501113b29 a6a33758b46e2689d9dd9ac69b57ec30c871fc066a70

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

***R.-*** Si, he recuperado la clave y resulta que son diferentes los mensajes encriptados por el fenómeno de aleatoriedad que, en esencia sucede cada que volvamos a cifrar el texto debido a que vamos a tener un nuevo padding producto de un valor aleatorio llamado "SEED" que genera un nuevo “padding” para el texto cifrado cada que ejecutemos de nueva cuenta el cifrado.

**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:E2CFF885901B3449E9C448BA5B948A8C4EE322152B3F1ACFA0148FB3A42

6DB74

Nonce:9Yccn/f5nJJhAt2S

¿Qué estamos haciendo mal?

En esencia el nonce el problema lo tenemos en el nonce porque este no satisface los requerimientos de aleatoriedad necesarios, Por definición debemos de generar este nonce de manera aleatoria para que así sea mucho más difícil para un actor malicioso lograr atacarnos.

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.

Texto cifrado en hexadecimal:

texto cifrado en hexadecimal:

5dcbb6261d0fba29ce39431e9a013b34cbca2a4e04bb2d90149d61f4afd04d65e2abdd9d84bba6 eb8307095f5078fbfc16256d

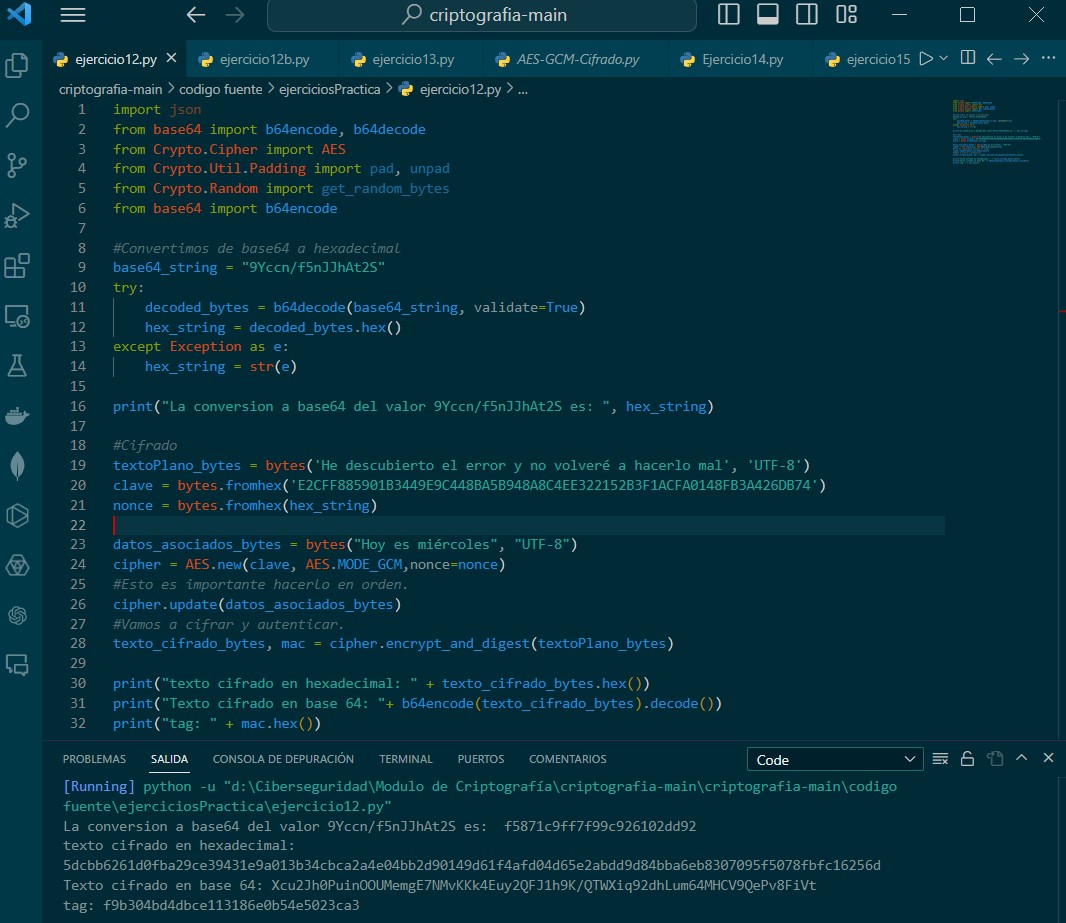
Texto cifrado en base64:

Texto cifrado en base 64:

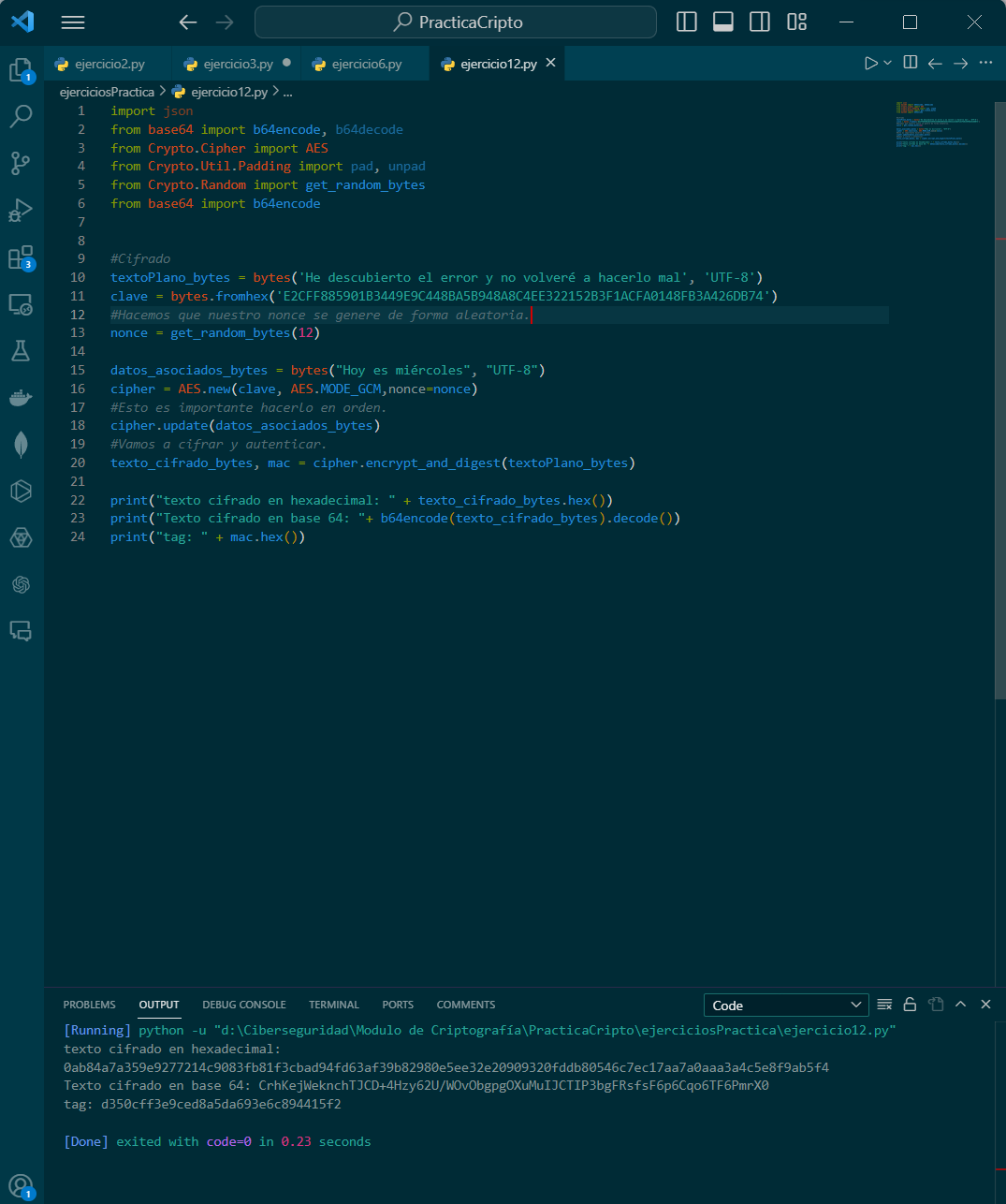
Xcu2Jh0PuinOOUMemgE7NMvKKk4Euy2QFJ1h9K/QTWXiq92dhLum64MHCV9QePv8FiVt

nonce en hexadecimal:

f5871c9ff7f99c926102dd92



***Descifrado para comprobar que las claves en hexadecimal y el nonce aleatorio fueron bien logradas:***

******

**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.

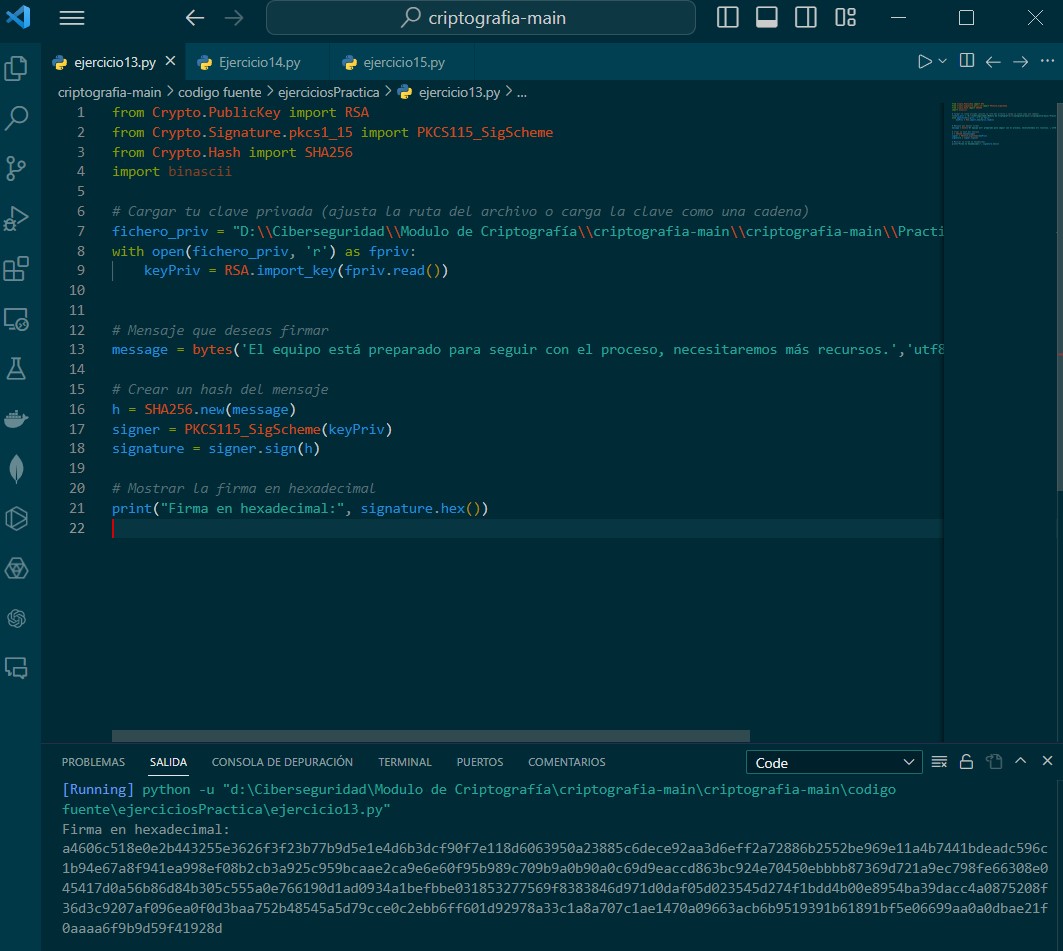
¿Cuál es el valor de la firma en hexadecimal?

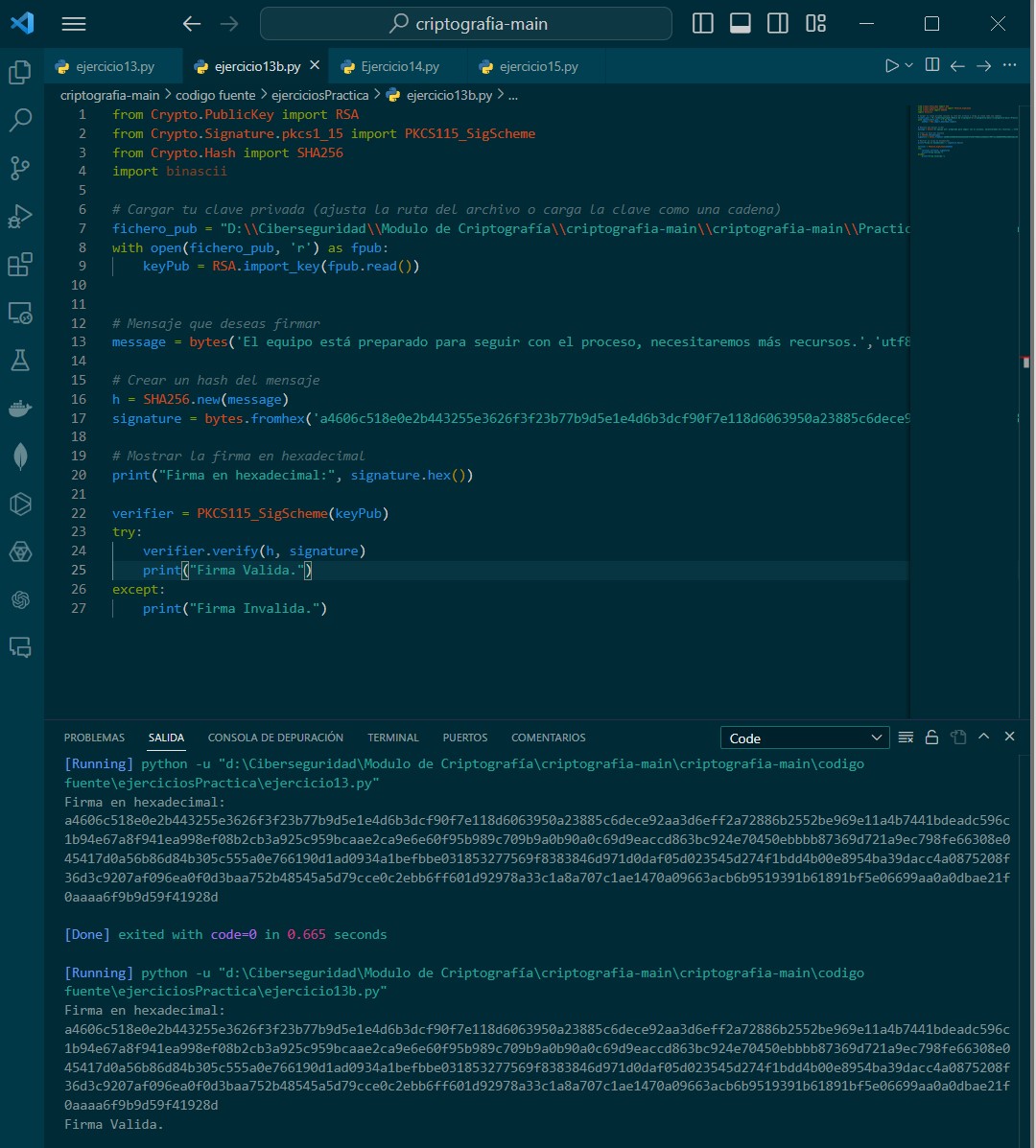
***R.-***

a4606c518e0e2b443255e3626f3f23b77b9d5e1e4d6b3dcf90f7e118d6063950a23885c6dece92aa 3d6eff2a72886b2552be969e11a4b7441bdeadc596c1b94e67a8f941ea998ef08b2cb3a925c959bc aae2ca9e6e60f95b989c709b9a0b90a0c69d9eaccd863bc924e70450ebbbb87369d721a9ec798fe 66308e045417d0a56b86d84b305c555a0e766190d1ad0934a1befbbe031853277569f8383846d9

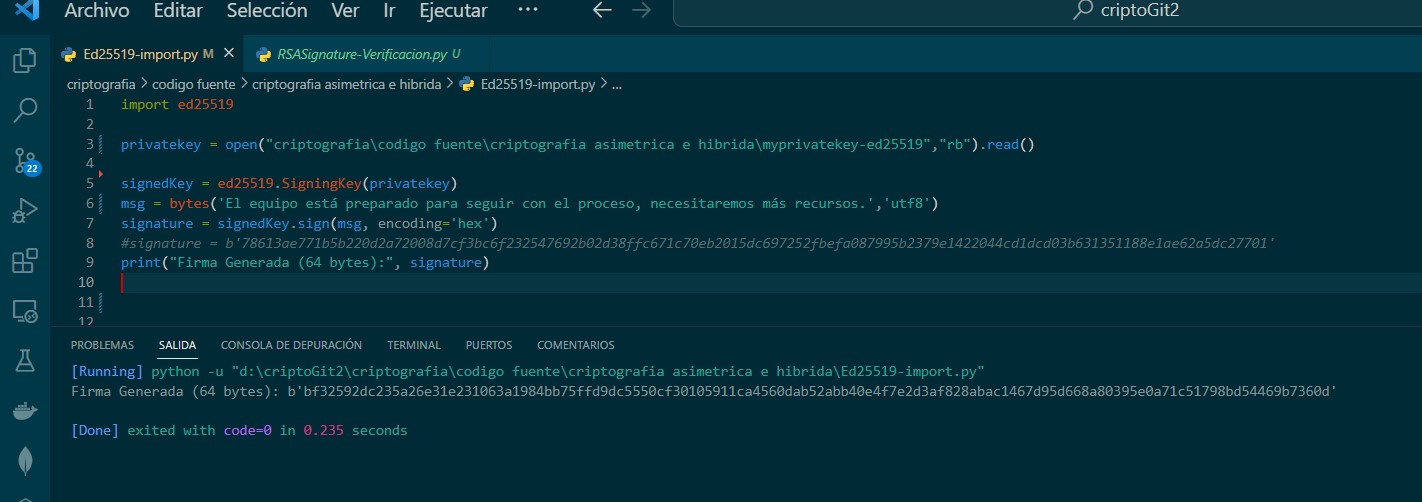
71d0daf05d023545d274f1bdd4b00e8954ba39dacc4a0875208f36d3c9207af096ea0f0d3baa752b

48545a5d79cce0c2ebb6ff601d92978a33c1a8a707c1ae1470a09663acb6b9519391b61891bf5e06 699aa0a0dbae21f0aaaa6f9b9d59f41928d





Calcula la firma (en hexadecimal) con la curva elíptica ed25519, usando las claves ed25519priv y ed25519-publ.

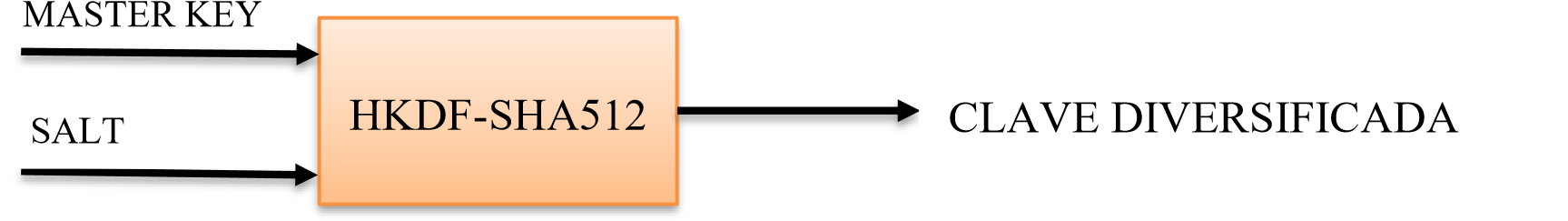


***R.-*** Firma Generada (64 bytes):

bf32592dc235a26e31e231063a1984bb75ffd9dc5550cf30105911ca4560dab52abb40e4f7e2d3af 828abac1467d95d668a80395e0a71c51798bd54469b7360d

**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?

La clave maestra es

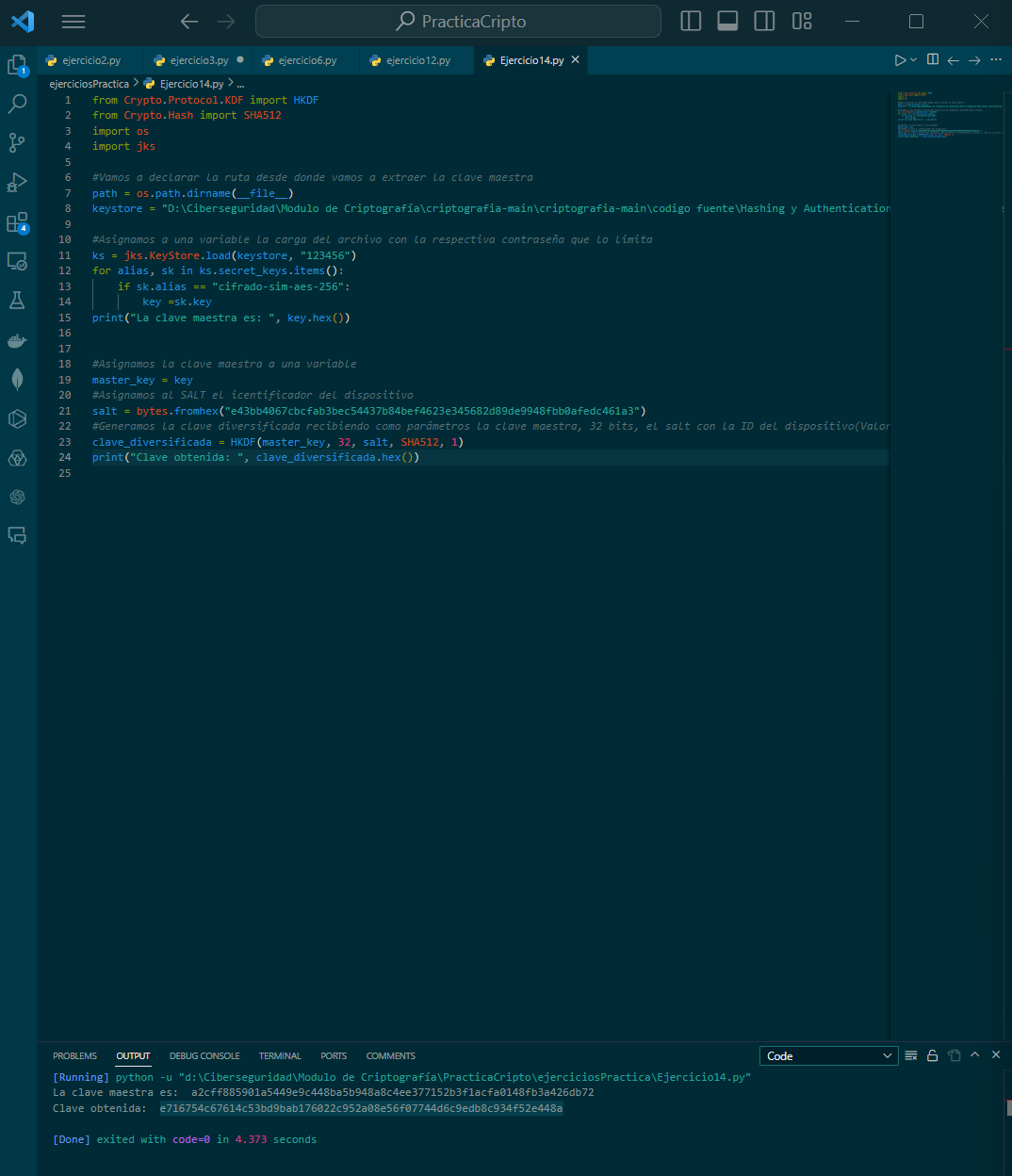
a2cff885901a5449e9c448ba5b948a8c4ee377152b3f1acfa0148fb3a426db72

Y

Clave Diversificada

es: e716754c67614c53bd9bab176022c952a08e56f07744d6c9edb8c934f52e448a

En este caso en particular hemos utilizado el SALT como elemento diversificador, pasandole la clave en hexadecimal para realizar el proceso de obtener nuestra clave diversificada, un SHA512 le acompaña.



**15.** Nos envían un bloque TR31:

D0144D0AB00S000042766B9265B2DF93AE6E29B58135B77A2F616C8D515ACDB

E6A5626F79FA7B4071E9EE1423C6D7970FA2B965D18B23922B5B2E5657495E0

3CD857FD37018E111B

Donde la clave de transporte para desenvolver (unwrap) el bloque es:

A1A10101010101010101010101010102

¿Con qué algoritmo se ha protegido el bloque de clave? ***R.-*** Con un algoritmo de cifrado ‘AES’

¿Para qué algoritmo se ha definido la clave? ***R.-***Para Un algoritmo de cifrado ‘AES’

¿Para qué modo de uso se ha generado? ***R.-*** Nos permite cifrar y descifrar

¿Es exportable? ***R.-*** Si, Si es exportable

¿Para qué se puede usar la clave? ¿Qué valor tiene la clave? ***R.-*** Clave de cifrado simétrico para encriptar datos

