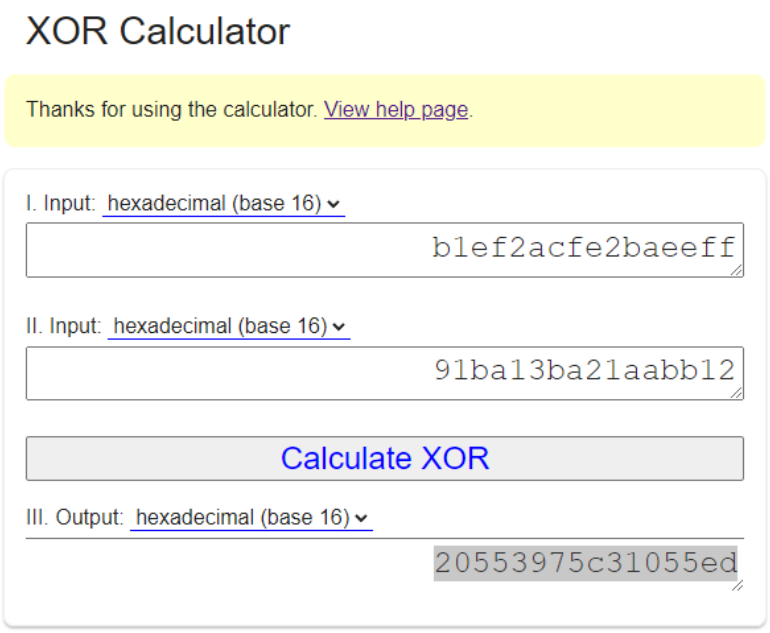
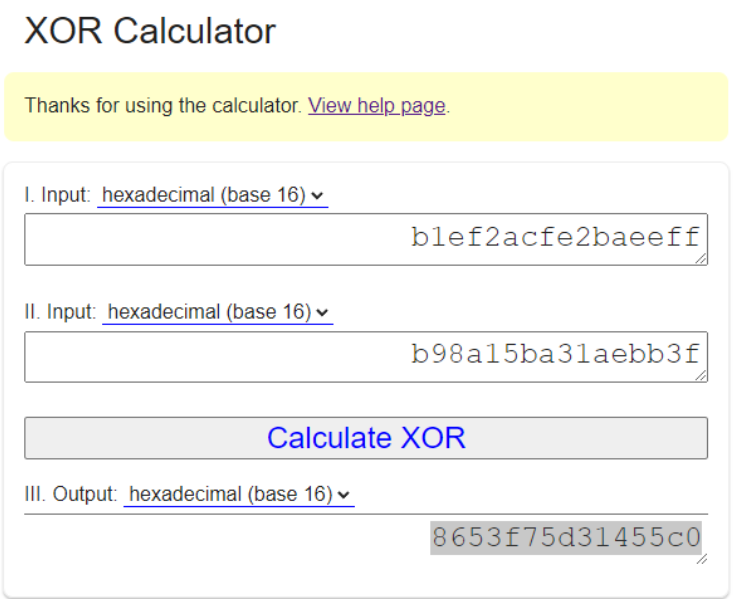
Álvaro García De La Mata

PRÁCTICA: CRIPTOGRAFÍA

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.
    a. **La solución en ambos casos es realizar un XOR, para ello vamos a pasar las 2 claves que tenemos a binario para realizar el xor y luego el resultado le retornamos a hexadecimal (Existen un montón de páginas para pasar de un sistema a otro o que simplemente hacen un xor directamente)**
    b. La clave fija en código es B1EF2ACFE2BAEEFF, mientras que en desarrollo sabemos que la clave final (en memoria) es 91BA13BA21AABB12.
    c. ¿Qué valor ha puesto el Key Manager en properties para forzar dicha clave final?
        i. **20553975c31055ed**
        ii. 

    d. 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.
    e. ¿Qué clave será con la que se trabaje en memoria?
        i. **8653f75d31455c0**

ii.



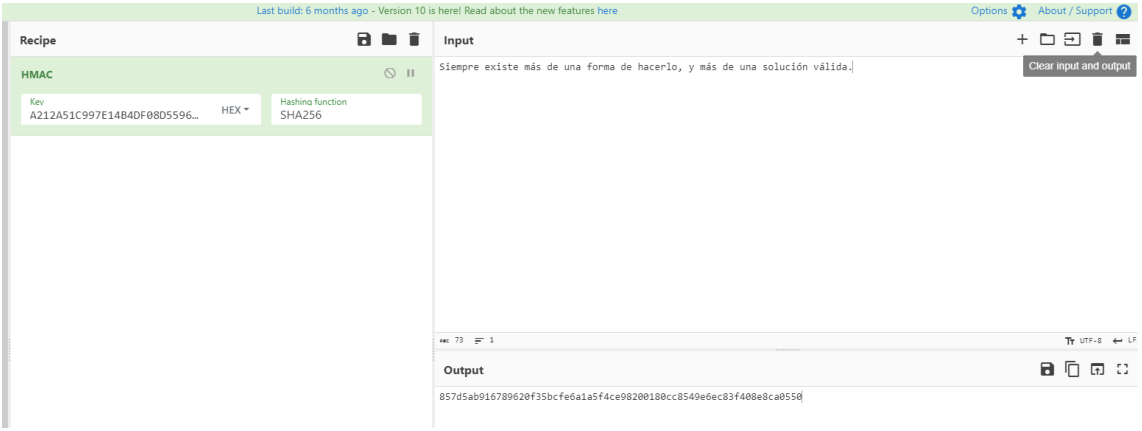
2. Dada la clave con etiqueta “cifrado-sim-aes-256” que contiene el keystore. El iv estará compuesto por el hexadecimal correspondiente a ceros binarios (“00”). Se requiere obtener el dato en claro correspondiente al siguiente dato cifrado: TQ9SOMKc6aFS9SlxhfK9wT18UXpPCd505Xf5J/5nLI7Of/o0QKIWXg3nu1RRz4QWElezdrLAD5LO4USt3aB/i50nvvJbBiG+le1ZhpR84oI=
    a. Para este caso, se ha usado un AES/CBC/PKCS7. Si lo desciframos, ¿qué obtenemos?
        i. El texto en claro es: Esto es un cifrado en bloque típico. Recuerda, vas por el buen camino. Ánimo.
    b. ¿Qué ocurre si decidimos cambiar el padding a x923 en el descifrado?
        i. En este caso, el texto en claro original tiene una longitud que requiere la adición de un solo byte de relleno. Al descifrar utilizando el esquema de relleno x923, este byte de relleno se elimina durante el proceso de descifrado.
    c. ¿Cuánto padding se ha añadido en el cifrado?
        i. 01
    d. Se valorará positivamente, obtener el dato de la clave desde el keystore mediante codificación en Python (u otro lenguaje).
        i. He dejado el código (AES-CBC-descifrado(Ej2).py)
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.
    a. ¿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.
        i. Para mejorar el sistema de una forma sencilla garantizando la confidencialidad y la integridad es utilizando un algoritmo sucesor de chacha20 → ChaCha20_Poly1305
        ii. nonce = f5871c9ff7f99c926102dd92

iii. Mensaje cifrado en HEX = 4ec95921ca8b757e2336605c7dbab8f4d40b5b4d220e66aa978f740d3e59b0cd70e3217242991cc140bb1e1a
iv. Mensaje cifrado en B64 = TslZIcqLdX4jNmBcfbq49NQLW00iDmaql490DT5ZsM1w4yFyQpkcwUC7Hho=
v. He dejado el código (ChaCha20_Poly1305-Cifrado(Ej3).py)

4. Tenemos el siguiente jwt, cuya clave es “Con KeepCoding aprendemos”
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmlvIjoiRG9uIFBlcGl0byBkZSBsb3MgcGFsb3RlcyIsInJvbCI6ImlzTm9ybWFsIiwiaWF0IjoxNjY3OTMzNTMzfQ.gfhw0dDxp6oixMLXXRP97W4TDTrv0y7B5YjD0U8ixrE
    a. En https://jwt.io/ podemos sacar información del JWT.
    b. ¿Qué algoritmo de firma hemos realizado?
        i. El algoritmo de firma utilizado es "HS256".
    c. ¿Cuál es el body del jwt?
        i. {"usuario": "Don Pepito de los palotes","rol": "isNormal", "iat": 1667933533}
    d. Un hacker está enviando a nuestro sistema el siguiente jwt: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmlvIjoiRG9uIFBlcGl0byBkZSBsb3MgcGFsb3RlcyIsInJvbCI6ImlzQWRtaW4iLCJpYXQiOjE2Njc5MzM1MzN9.krgBkzCBQ5WZ8JnZHuRvmnAZdg4ZMeRNv2CIAODlHRI
    e. ¿Qué está intentando realizar?
        i. Está intentando cambiar el mensaje a: {'usuario': 'Don Pelpito de los palotes', 'rol': 'isAdmin', 'iat': 1667933533}
    f. ¿Qué ocurre si intentamos validarlo con pyjwt?
        i. En este caso, al intentar validar el token del hacker con pyjwt, la validación fallará, ya que la clave secreta utilizada para firmar el token no coincide con la clave secreta "Con KeepCoding aprendemos". La firma no será válida y el código generará un error.

5. El siguiente hash se corresponde con un SHA3 Keccak del texto “En KeepCoding aprendemos cómo protegernos con criptografía”.
bced1be95fbd85d2ffcce9c85434d79aa26f24ce82fbd4439517ea3f072d5fe
    a. Ejercicio realizado con (Hash Cadenas(Ej5).py)
    b. ¿Qué tipo de SHA3 hemos generado?
        i. Un SHA3 de 256 bits, si haces uno de 224 por ejemplo se nota la diferencia de tamaño
    c. Y si hacemos un SHA2, y obtenemos el siguiente resultado: 4cec5a9f85dcc5c4c6ccb603d124cf1cdc6dfe836459551a1044f4f2908aa5d63739506f6468833d77c07cfd69c488823b8d858283f1d05877120e8c5351c833
    d. ¿Qué hash hemos realizado?
        i. Un SHA-256 porque la cadena tiene 64 caracteres hexadecimales. Si cada carácter hexadecimal representa 4bits y multiplicamos la longitud de la cadena por 4 obtenemos la longitud del hash en bit
        ii. 64 caracteres * 4 bits = 256 bits → SHA-256

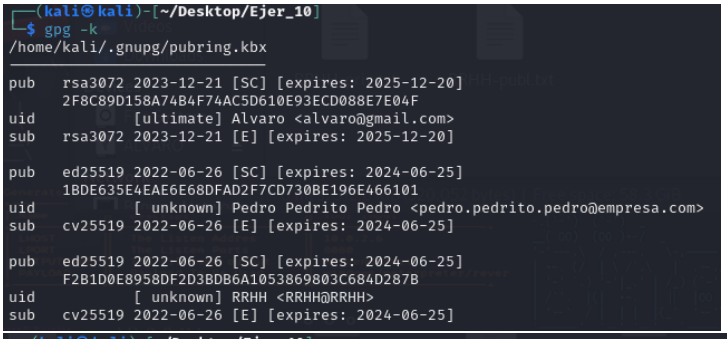
e. Genera ahora un SHA3 Keccak de 256 bits con el siguiente texto: “En KeepCoding aprendemos cómo protegernos con criptografía.” ¿Qué propiedad destacarías del hash, atendiendo a los resultados anteriores?
   i. bced1be95fbd85d2ffcce9c85434d79aa26f24ce82fbd4439517ea3f072d56fe
   ii. bced1be95fbd85d2ffcce9c85434d79aa26f24ce82fbd4439517ea3f072d5fe
   iii. La diferencia es clara (un 6), ya que la diferencia de los dos textos es el punto al final en el que uno lo lleva y otro no lo lleva.

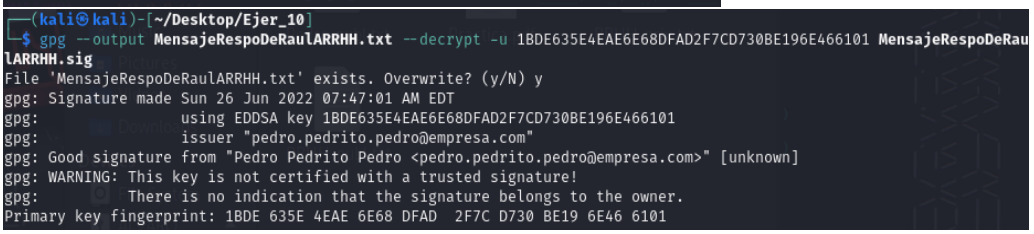
6. Calcula el hmac-256 (usando la clave contenida en el Keystore) del siguiente texto: Siempre existe más de una forma de hacerlo, y más de una solución válida.
   a. Se debe evidenciar la respuesta. Cuidado si se usan herramientas fuera de los lenguajes de programación, por las codificaciones es mejor trabajar en hexadecimal.
   b. Hmac a través de código Python (Ejercicio_6.py) y Ciberchef: 857d5ab916789620f35bcfe6a1a5f4ce98200180cc8549e6ec83f408e8ca0550
   c. 

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.
   a. ¿Por qué crees que es una mala opción?
      i. Porque SHA-1 actualmente se encuentra obsoleto debido a las vulnerabilidades que se han encontrado en él. Recordemos que su longitud es de 160 bits lo que le hace menos seguro que un SHA-256 (256bits) o un SHA-3 con una la longitud de bits variable
   b. 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?
      i. Añadir un algoritmo de derivación de claves como bcrypt, scrypt o Argon2(Siendo esta la mejor opción entre las 3)
   c. Parece que el responsable se ha quedado conforme, tras mejorar la propuesta del SHA-256, no obstante, hay margen de mejora. ¿Qué propondrías?
      i. El siguiente paso para mejorar aún más la seguridad sería optar por un HASH SHA-3, un hash similar al SHA-256 pero basado en una nueva estructura llamada “esponja”.

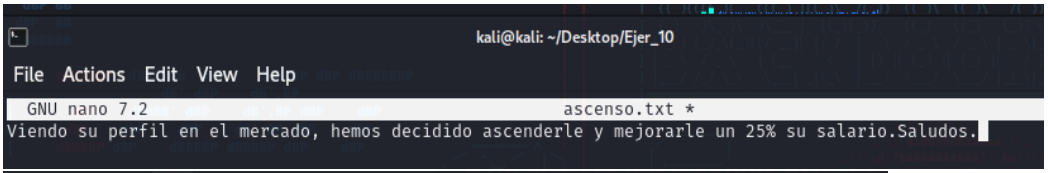
8. Tenemos la siguiente API REST, muy simple
    a. Como se puede ver en el API, tenemos ciertos parámetros que deben mantenerse confidenciales. Así mismo, nos gustaría que nadie nos modificase el mensaje sin que nos enterásemos. Se requiere una redefinición de dicha API para garantizar la integridad y la confidencialidad de los mensajes. Se debe asumir que el sistema end to end no usa TLS entre todos los puntos.
    b. ¿Qué algoritmos usarías?
        i. Los algoritmos que utilizaría que cubrirían nuestra necesidad (confidencialidad e integridad) serían:
            1. ChaCha20_Poly1305: Cubre la parte de confidencialidad cifrando los datos y la integridad a través de un MAC, tanto el emisor como el receptor comparten una clave secreta y el receptor al verificar el código de autentificación se puede detectar si se ha modificado el mensaje.
            2. CTR+MAC es otra opción válida ya que combinamos la confidencialidad del cifrado ctr más un HMAC que nos ofrece es integridad que buscamos.
            3. GCM, sería la evolución de CTR+MAC ya que es un cifrado que ofrece integridad (cifrado autenticado). Gracias a la utilización de un HASH nos permite tener esa integridad que buscamos.
            4. Para resumir que es lo que necesitaríamos para garantizar tanto la confidencialidad como la integridad sería un buen algoritmo de cifrado y un Hash o una firma digital (como RSA) para garantizar la integridad.

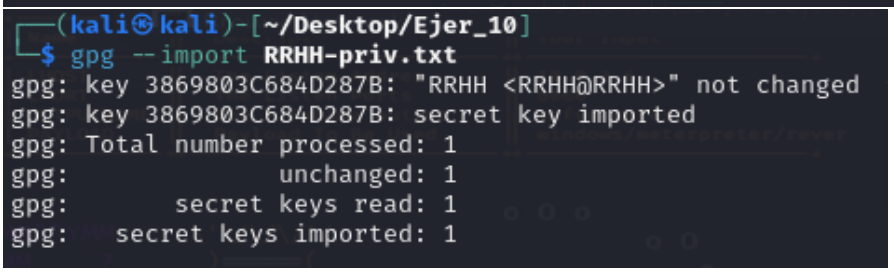
9. Se requiere calcular el KCV de la siguiente clave AES: A2CFF885901A5449E9C448BA5B948A8C4EE377152B3F1ACFA0148FB3A426DB72. Para lo cual, vamos a requerir el KCV(SHA-256) así como el KCV(AES). El KCV(SHA-256) se corresponderá con los 3 primeros bytes del SHA-256. Mientras que el KCV(AES) se corresponderá 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.
    a. Evidencia con código en: (kcv(Ej9).py)
    b. KCV AES: 5244db
    c. KCV SHA256: 95cbd2
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).
    a. Todos los archivos como prueba de mi trabajo en este ejercicio se encuentran en el directorio: (Ejer_10)
    b. 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.

i.


ii.


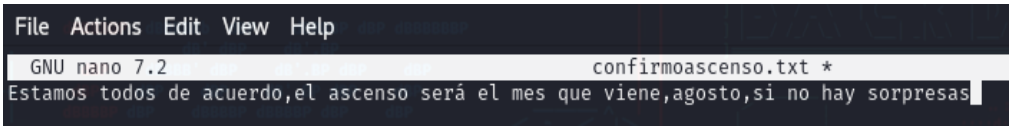
c. Así mismo, se requiere firmar el siguiente mensaje con la clave correspondiente de las anteriores, simulando que eres personal de RRHH: <u>Viendo su perfil en el mercado, hemos decidido ascenderle y mejorarle un 25% su salario. Saludos.</u>

i.


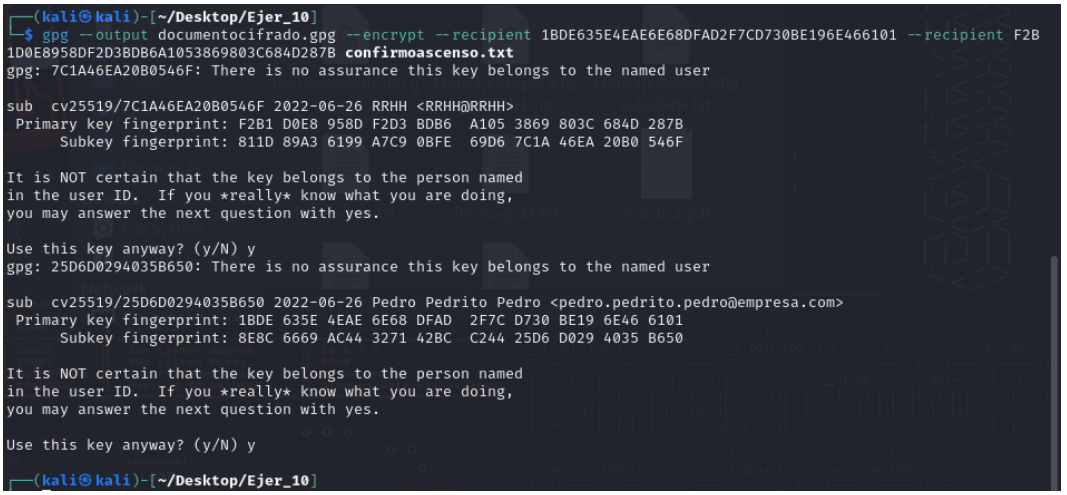
ii.


iii.


d. Por último, cifra el siguiente mensaje tanto con la clave pública de RRHH como la de Pedro y adjunta el fichero con la práctica: <u>Estamos todos de acuerdo, el ascenso será el mes que viene, agosto, si no hay sorpresas</u>

i.


ii.



11. 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.

    a. El texto cifrado es el siguiente:

    b. b72e6fd48155f565dd2684df3ffa8746d649b11f0ed4637fc4c99d18283b32e1709b30c96b4a8a20d5dbc639e9d83a53681e6d96f76a0e4c279f0dffa76a329d04e3d3d4ad629793eb00cc76d10fc00475eb76bfbc1273303882609957c4c0ae2c4f5ba670a4126f2f14a9f4b6f41aa2edba01b4bd586624659fca82f5b4970186502de8624071be78ccef573d896b8eac86f5d43ca7b10b59be4acf8f8e0498a455da04f67d3f98b4cd907f27639f4b1df3c50e05d5bf63768088226e2a9177485c54f72407fdf358fe64479677d8296ad38c6f177ea7cb74927651cf24b01dee27895d4f05fb5c161957845cd1b5848ed64ed3b03722b21a526a6e447cb8ee

    c. 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?

        i. El esquema de relleno Optimal Asymmetric Encryption Padding (OAEP) utiliza funciones de hash y operaciones aleatorias para garantizar la seguridad del cifrado y para evitar problemas de seguridad. Aunque cifres el mismo mensaje con la misma clave pública varias veces, obtendrás diferentes textos cifrados.
        ii. Todos los archivos como prueba de mi trabajo en este ejercicio se encuentran en el directorio: (Ejer_11)

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:
    a. El archivo .py que he utilizado es (AES-GCM-Cifrado(EJ12).py)
    b. Key: E2CFF885901B3449E9C448BA5B948A8C4EE322152B3F1ACFA0148FB3A426DB74
    c. Nonce: 9Yccn/f5nJJhAt2S

d. ¿Qué estamos haciendo mal?
    i. En GCM a la hora de cifrar necesitamos definir una cadena de datos asociados, dato que no se ha proporcionado en el enunciado de este ejercicio. Un tag en GCM proporciona integridad garantizando así que los datos no han sido manipulados.
    ii. El que yo he elegido es: "felipe,profesor de keepcoding"
e. Cifra el siguiente texto: **He descubierto el error y no volveré a hacerlo mal**
f. Usando para ello, la clave, y el nonce indicados. El texto cifrado preséntalo en hexadecimal y en base64.
    i. Texto cifrado en hexadecimal: 805ca6333de1d5d453193f2f2623ccef8dddd49bbde7183a3f6558d6f320faf78d9ff0efb6e75dc82b8c77234b61eaa29b1347561f04e0bca18f3ca1790d3444
    ii. Tag en hexadecimal: a52e1b9857ead0e4994354aeaf81d268
    iii. Texto cifrado en base64: gFymMz3h1dRTGT8vJiPM743d1Ju95xg6P2VY1vMg+veNn/DvtuddyCuMdyNLYeqimxNHVh8E4LyhjzyheQ00RA==
    iv. Tag en base64: pS4bmFfq0OSZQ1Sur4HSaA==

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:
    a. Este ejercicio le he realizado en un entorno virtualizado (Kali_linux_2023.4) a través del documento (Ed25519-Signature-import.py)
    b. El equipo está preparado para seguir con el proceso, necesitaremos más recursos. ¿Cuál es el valor de la firma en hexadecimal?
        i. Firma: 49718237962f213626a04ae2463e2a620aee13fee813e020758c9e40f2e3750982031e3013faa6ab71b5aa13b52a326966d693ed15dff34219ddae0dc4262bf0c92dafdd3899819c9a4e3db08b876de91798fbab0cf36e5dd0e369f6adea4a5699c48f48d79bb70277633b486ed0bc5f801824236b096cbcd839f581fa460abfb6033434df93d874e59434521396491d0b1f540dde1b5fe32e6ba13887eb2cc580b5e7d18d7eba4032c0e05e67688a7bfb9fc938465f60e1405471ec7ea0d9fd42b0661f4fcc2870aec7ba3885528ac942b7585875c03e8de52072098268ec2bd14d2cf051e85da84b862f75c0d34f0fb52fb0edf2023f84273415b7880ba766
    c. Calcula la firma (en hexadecimal) con la curva elíptica ed25519, usando las claves ed25519- priv y ed25519-publ.
        i.
```
import ed25519


privatekey = open("ed25519-priv","rb").read()

signedKey = ed25519.SigningKey(privatekey)
msg = bytes('Firmamos esto con la curva 25519','utf8')
signature = signedKey.sign(msg, encoding='hex')

print("Firma Generada (64 bytes):", signature)
```
        ii.
```
Firma Generada (64 bytes): b'470434f69bb45c7772bc64bc164081bbce669e5c24b98d5b3f77b903ecb321d9a12af9aa09f9ffae6e4732612e7e09e03772be03fc4025f6b485c0d7c9f8f80d'
```

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
    a. Ejercicio realizado a través de: (HKDF(Ej14).py)
    b. ¿Qué clave se ha obtenido?
        i. Clave key1: 3343138e24878b059aa59014f71ca6252976c6aa08be7642e16276e9819798e8
15. Nos envían un bloque TR31:
    a. D0144D0AB00S000042766B9265B2DF93AE6E29B58135B77A2F616C8D515ACDB E6A5626F79FA7B4071E9EE1423C6D7970FA2B965D18B23922B5B2E5657495E0 3CD857FD37018E111B Donde la clave de transporte para desenvolver (unwrap) el bloque es: A1A10101010101010101010101010102
    b. Ejercicio realizado gracias a este archivo: (cabeceraTR31(Ej15).py) y esta documentación: #https://github.com/knovichikhin/psec/blob/master/psec/tr31.py
    c. ¿Con qué algoritmo se ha protegido el bloque de clave?
        i. AES
    d. ¿Para qué algoritmo se ha definido la clave?
        i. AES
    e. ¿Para qué modo de uso se ha generado?
        i. Tanto cifrar como descifrar/envolver y desenvolver
    f. ¿Es exportable?
        i. i. Es Sensible, exportable bajo una clave no confiable
    g. ¿Para qué se puede usar la clave?
        i. i. Clave simétrica para cifrado de datos
    h. ¿Qué valor tiene la clave?
        i. En Hexadecimal: c1c1c1c1c1c1c1c1c1c1c1c1c1c1c1c1