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.

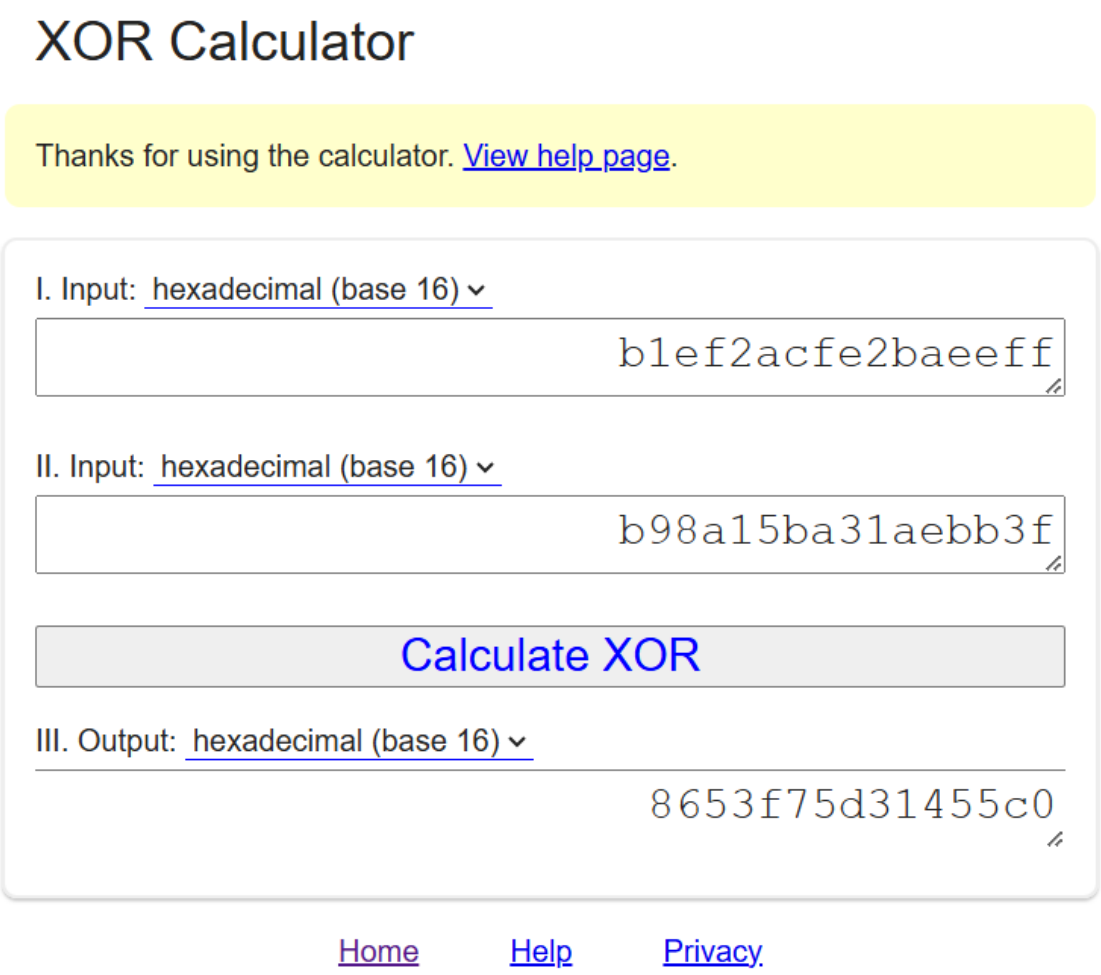
El objetivo principal de este módulo es adquirir conocimientos de criptografía y por ello se considera fundamental usar cualquier herramienta que pueda ayudar a su resolución, demostrando que no sólo se obtiene el dato sino que se tiene un conocimiento profundo del mismo. Si durante la misma no se indica 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?

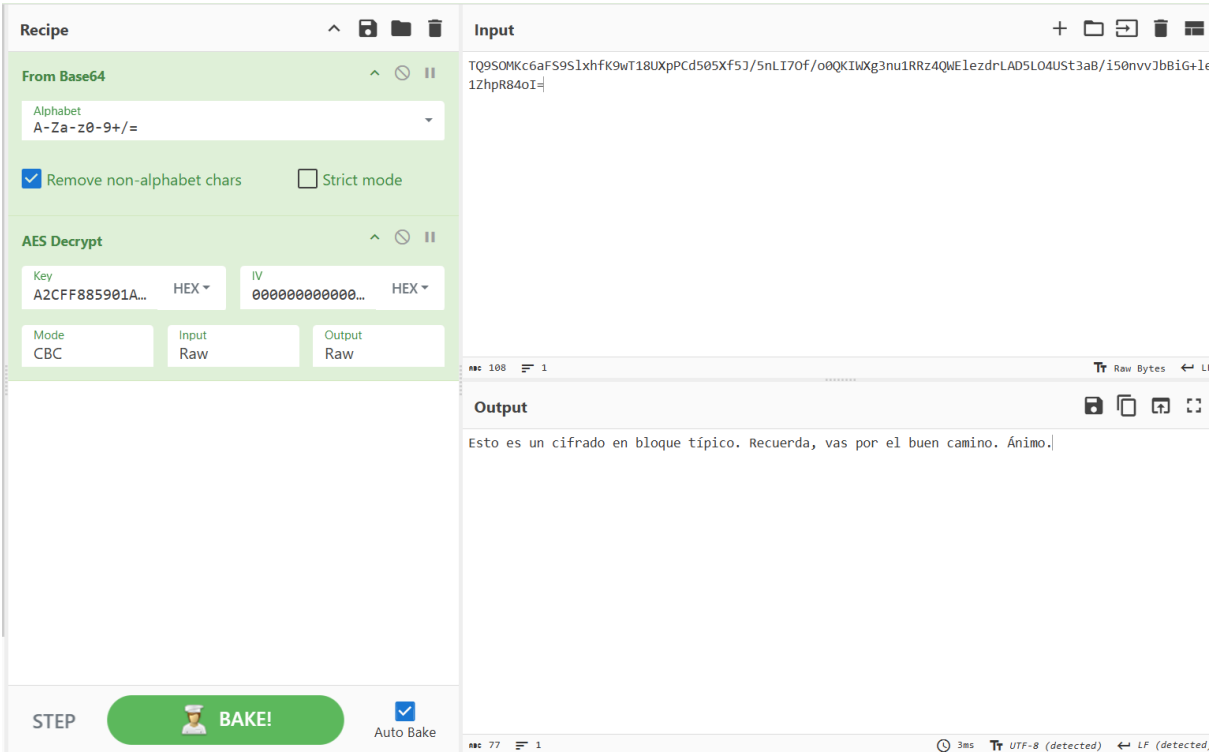
   La clave fija, recordemos es B1EF2ACFE2BAEEFF, mientras que en producción sabemos que la parte dinámica que se modifica en los ficheros de propiedades es B98A15BA31AEBB3F. ¿Qué clave será con la que se trabaje en memoria?



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+le1ZhpR84oI=

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



¿Qué ocurre si decidimos cambiar el padding a x923 en el descifrado?
The output message would likely come out incorrect.
¿Cuánto padding se ha añadido en el cifrado?

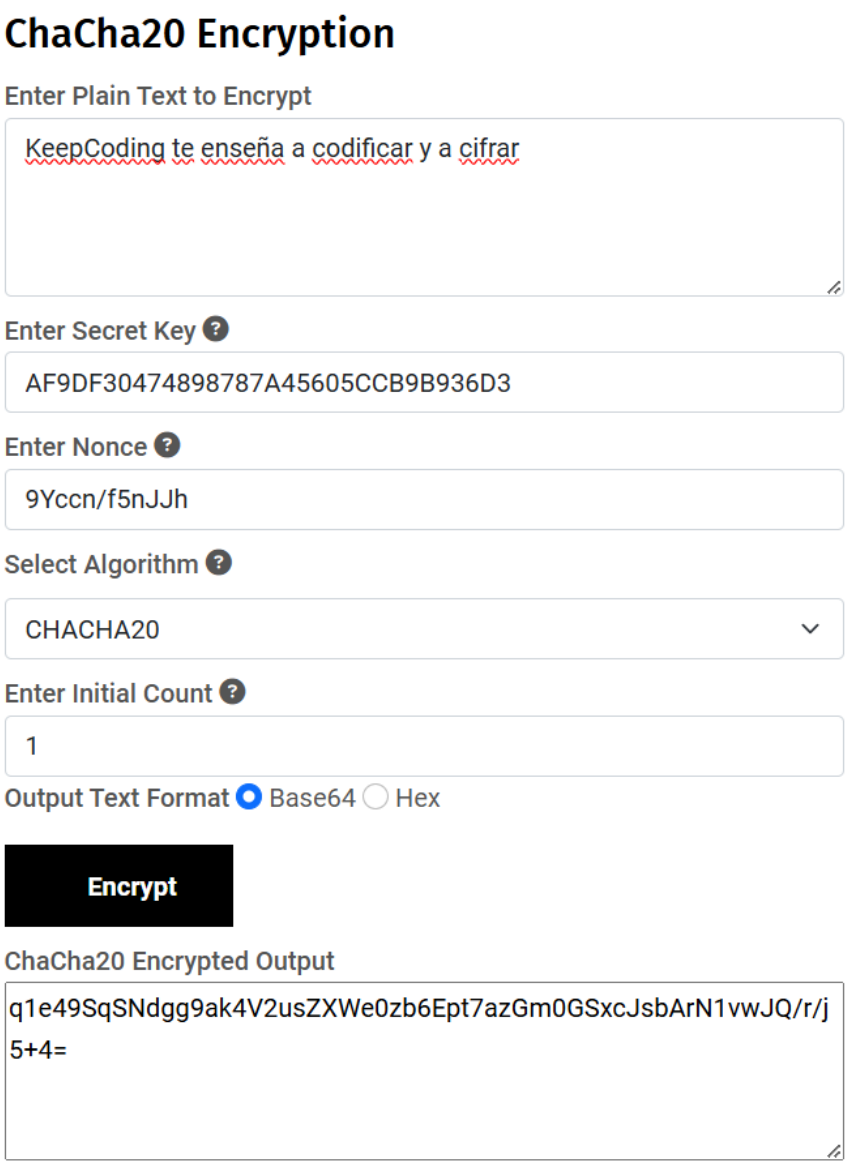
The padding X9.23 adds 01000000

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

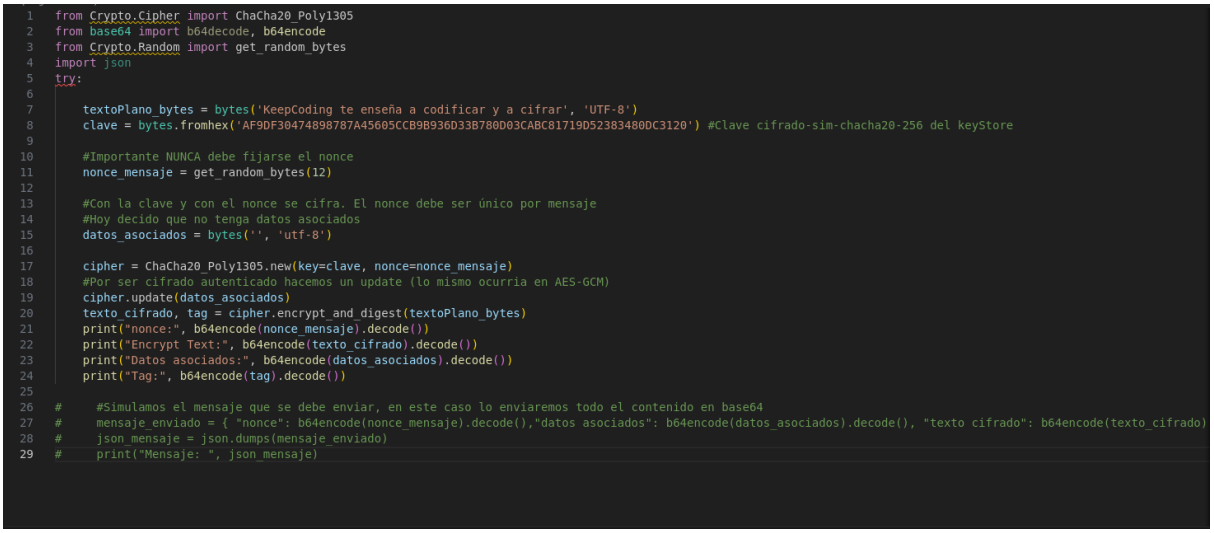
3. Se requiere cifrar el texto “KeepCoding te enseña a codificar y a cifrar”. La clave para ello, tiene la etiqueta en el Keystore “cifrado-sim-chacha20-256”. El nonce “9Yccn/f5nJJhAt2S”. El algoritmo que se debe usar es un Chacha20.

¿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.



To improve the integrity and confidentiality of the system, it's recommended to use ChaCha20-Poly1305. Poly1305 is what actually makes sure the data isn't tampered with, and has integrity. The following screenshot is the improved upon code.



```python
from Crypto.Cipher import ChaCha20_Poly1305
from base64 import b64decode, b64encode
from Crypto.Random import get_random_bytes
import json
try:

    textoPlano_bytes = bytes('KeepCoding te enseña a codificar y a cifrar', 'UTF-8')
    clave = bytes.fromhex('AF9DF30474898787A45605CCB9B936D33B780D03CABC81719D52383480DC3120') #Clave cifrado-sim-chacha20-256 del keyStore

    #Importante NUNCA debe fijarse el nonce
    nonce_mensaje = get_random_bytes(12)

    #Con la clave y con el nonce se cifra. El nonce debe ser único por mensaje
    #Hoy decido que no tenga datos asociados
    datos_asociados = bytes('', 'utf-8')

    cipher = ChaCha20_Poly1305.new(key=clave, nonce=nonce_mensaje)
    #Por ser cifrado autenticado hacemos un update (lo mismo ocurria en AES-GCM)
    cipher.update(datos_asociados)
    texto_cifrado, tag = cipher.encrypt_and_digest(textoPlano_bytes)
    print("nonce:", b64encode(nonce_mensaje).decode())
    print("Encrypt Text:", b64encode(texto_cifrado).decode())
    print("Datos asociados:", b64encode(datos_asociados).decode())
    print("Tag:", b64encode(tag).decode())

#     #Simulamos el mensaje que se debe enviar, en este caso lo enviaremos todo el contenido en base64
#     mensaje_enviado = { "nonce": b64encode(nonce_mensaje).decode(),"datos asociados": b64encode(datos_asociados).decode(), "texto cifrado": b64encode(texto_cifrado)
#     json_mensaje = json.dumps(mensaje_enviado)
#     print("Mensaje: ", json_mensaje)
```

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

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmlvIjoiRG9uIFBlcGl0byBkZSB

sb3MgcGFsb3RlcyIsInJvbCI6ImlzTm9ybWFsIiwiaWF0IjoxNjY3OTMzNTMzfQ.gfhw0

dDxp6oixMLXXRP97W4TDTrv0y7B5YjD0U8ixrE

¿Qué algoritmo de firma hemos realizado?
The algorithm is HMAC-SHA256
¿Cuál es el body del jwt?

```
{
 "usuario": "Don Pepito de los palotes",
 "rol": "isNormal",
 "iat": 1667933533
}
```

Un hacker está enviando a nuestro sistema el siguiente jwt:

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmlvIjoiRG9uIFBlcGl0byBk
ZSBsb3MgcGFsb3RlcyIsInJvbCI6ImlzQWRtaW4iLCJpYXQiOjE2Njc5MzM1MzN9
.krgBkzCBQ5WZ8JnZHuRvmnAZdg4ZMeRNv2CIAODlHRI hhhlhlhlhlkhlhhllkhl

¿Qué está intentando realizar?
The hacker is tampering with the code attempting to give themselves admin by changing one of the lines of code to “isNormal” to “isAdmin”.
¿Qué ocurre si intentamos validarlo con pyjwt?
If we try to validate the code, the validation will fail because the signature is invalid; Altering the payload will alter the hash, so it doesn’t match anymore with the given key.

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?

This is a SHA3-256 hash. It's due to the length being 64 characters which corresponds to 256 bits.

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

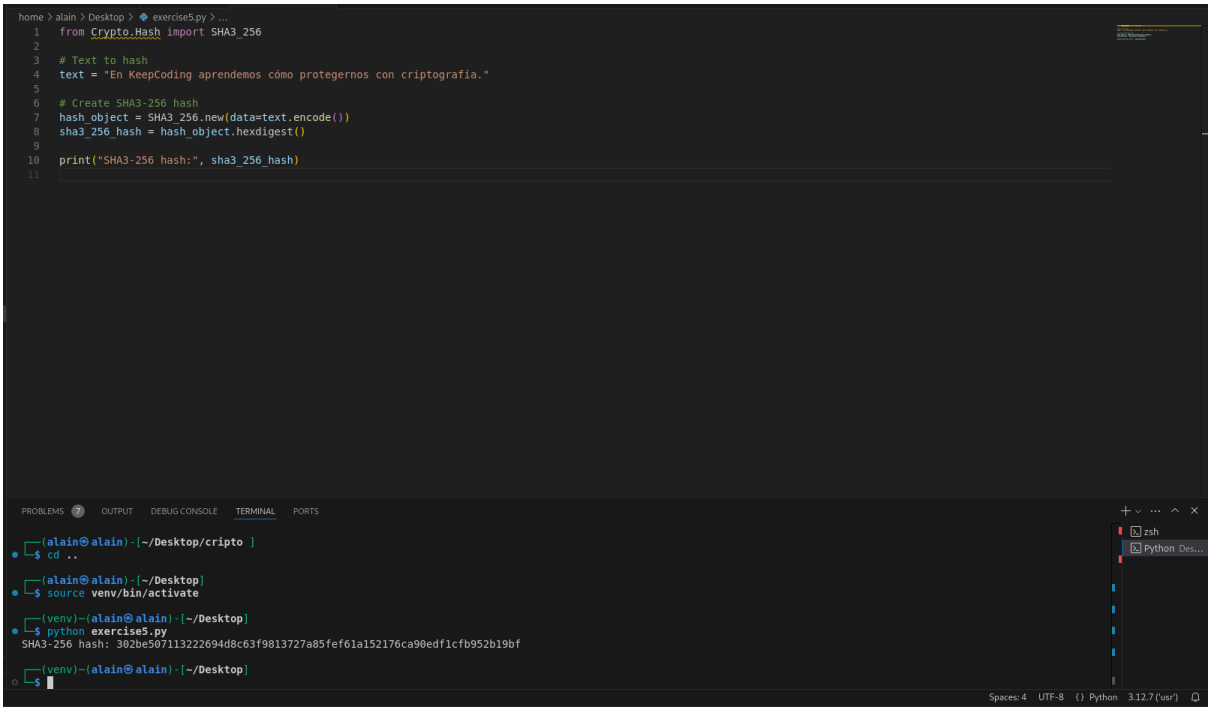
4cec5a9f85dcc5c4c6ccb603d124cf1cdc6dfe836459551a1044f4f2908aa5d63739506f

6468833d77c07cfd69c488823b8d858283f1d05877120e8c5351c833

¿Qué hash hemos realizado?

This hash would be SHA-512, because SHA-512 produces a 128-character hash.

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?

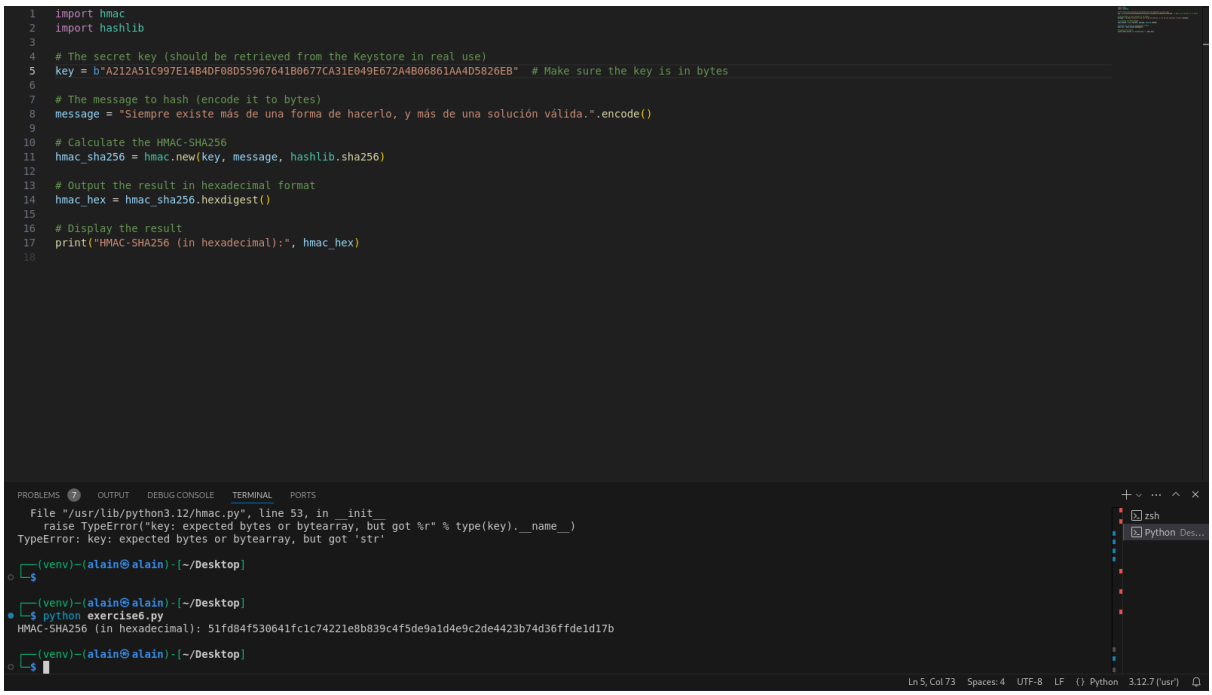


Adding just a period changed the entire hash, and this is due to the avalanche effect. Both hashing algorithms are part of different families (SHA3 vs SHA2), but they serve similar purposes of ensuring data integrity, though they operate differently internally.

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.



The result:
HMAC-SHA256 (in hexadecimal):
51fd84f530641fc1c74221e8b839c4f5de9a1d4e9c2de4423b74d36ffde1d17b

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?
SHA-1 is a bad option because it has security vulnerabilities. For some reason, two different inputs are able to output the same hash, causing a potential collision attack. In conclusion, it loses very easily to modern attack techniques.

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?

While SHA-256 is good, it can be improved upon by using salting. Salting would prevent two identical passwords from having the same hash.

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

there are better alternatives. The best option would be to use Argon2, because it's considered the most secure algorithm at the moment. Bcrypt and scrypt are also good options, but argon2 is the best.

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.

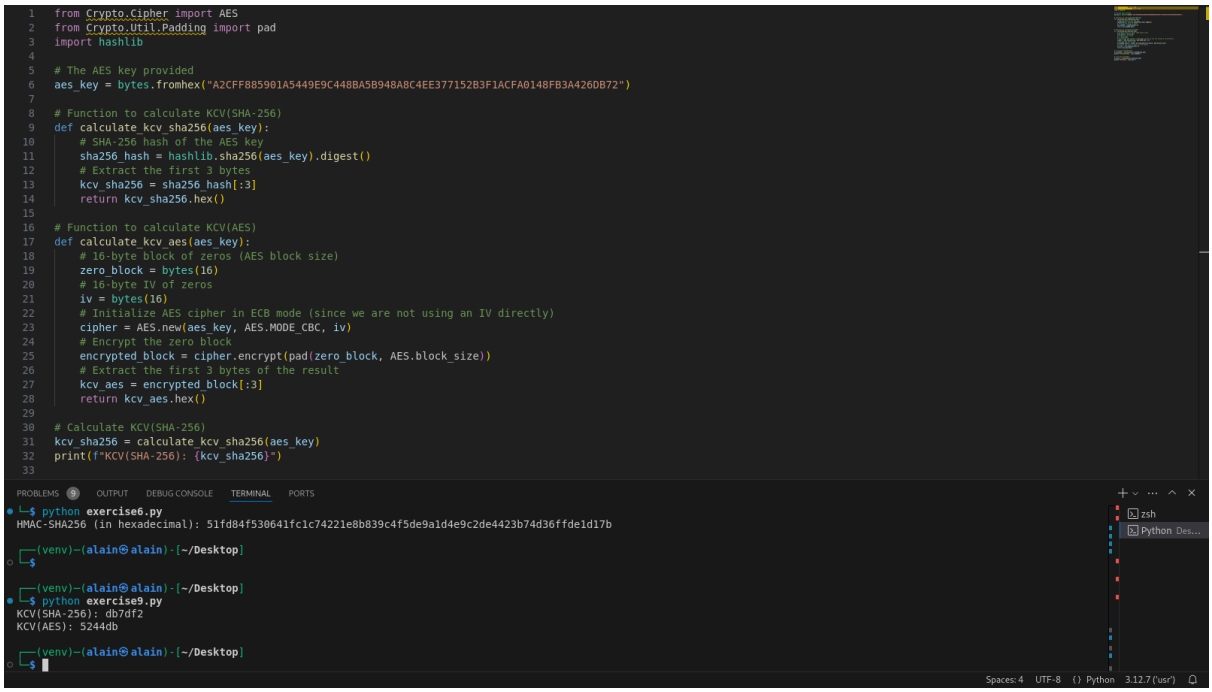
¿Qué algoritmos usarías?

It would be best to use AES for encryption; particularly for the parts relating to users' names and credit card numbers. After that, HMAC-SHA-256 would be good to use for integrity; to hash the message and verify it.

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.



The result:

KCV(SHA-256): db7df2

KCV(AES): 5244db

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.

```
┌──(alain㉿alain)-[~/Desktop/cripto/cripto-main/Practica]
└─$ gpg --verify MensajeRespoDeRaulARRHH.sig

gpg: Signature made Sun Jun 26 13:47:01 2022 CEST
gpg:                using EDDSA key 1BDE635E4EAE6E68DFAD2F7CD730BE196E466101
gpg:                issuer "pedro.pedrito.pedro@empresa.com"
gpg: Good signature from "Pedro Pedrito Pedro <pedro.pedrito.pedro@empresa.com>" [ultimate]
```

```
┌──(alain㉿alain)-[~/Desktop/cripto/cripto-main/Practica]
└─$ gpg --output MensajeRRHH.sig --sign MensajeRRHH.txt

File 'MensajeRRHH.sig' exists. Overwrite? (y/N) y

┌──(alain㉿alain)-[~/Desktop/cripto/cripto-main/Practica]
```

```
┌──(alain㉿alain)-[~/Desktop/cripto/cripto-main/Practica]
└─$ gpg --list-keys

/home/alain/.gnupg/pubring.kbx
------------------------------
pub   ed25519 2022-06-26 [SC]
      1BDE635E4EAE6E68DFAD2F7CD730BE196E466101
uid           [ultimate] Pedro Pedrito Pedro <pedro.pedrito.pedro@empresa.com>
sub   cv25519 2022-06-26 [E]

pub   ed25519 2022-06-26 [SC] [expires: 2026-12-31]
      F2B1D0E8958DF2D3BDB6A1053869803C684D287B
uid           [ultimate] RRHH <RRHH@RRHH>
sub   cv25519 2022-06-26 [E] [expires: 2026-12-31]


┌──(alain㉿alain)-[~/Desktop/cripto/cripto-main/Practica]
└─$ gpg --list-secret-keys

/home/alain/.gnupg/pubring.kbx
------------------------------
sec   ed25519 2022-06-26 [SC] [expires: 2026-12-31]
      F2B1D0E8958DF2D3BDB6A1053869803C684D287B
uid           [ultimate] RRHH <RRHH@RRHH>
ssb   cv25519 2022-06-26 [E] [expires: 2026-12-31]
```

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.

```
┌──(alain㉿alain)-[~/Desktop/cripto/cripto-main/Practica]
└─$ gpg --output MensajeConfidencial.txt.decrypted --decrypt MensajeConfidencial.txt.gpg

gpg: encrypted with 255-bit ECDH key, ID 25D6D0294035B650, created 2022-06-26
      "Pedro Pedrito Pedro <pedro.pedrito.pedro@empresa.com>"
gpg: encrypted with 255-bit ECDH key, ID 7C1A46EA20B0546F, created 2022-06-26
      "RRHH <RRHH@RRHH>"

┌──(alain㉿alain)-[~/Desktop/cripto/cripto-main/Practica]
└─$ cat MensajeConfidencial.txt.decrypted
Estamos todos de acuerdo, el ascenso será el mes que viene, agosto, si no hay sorpresas.
```

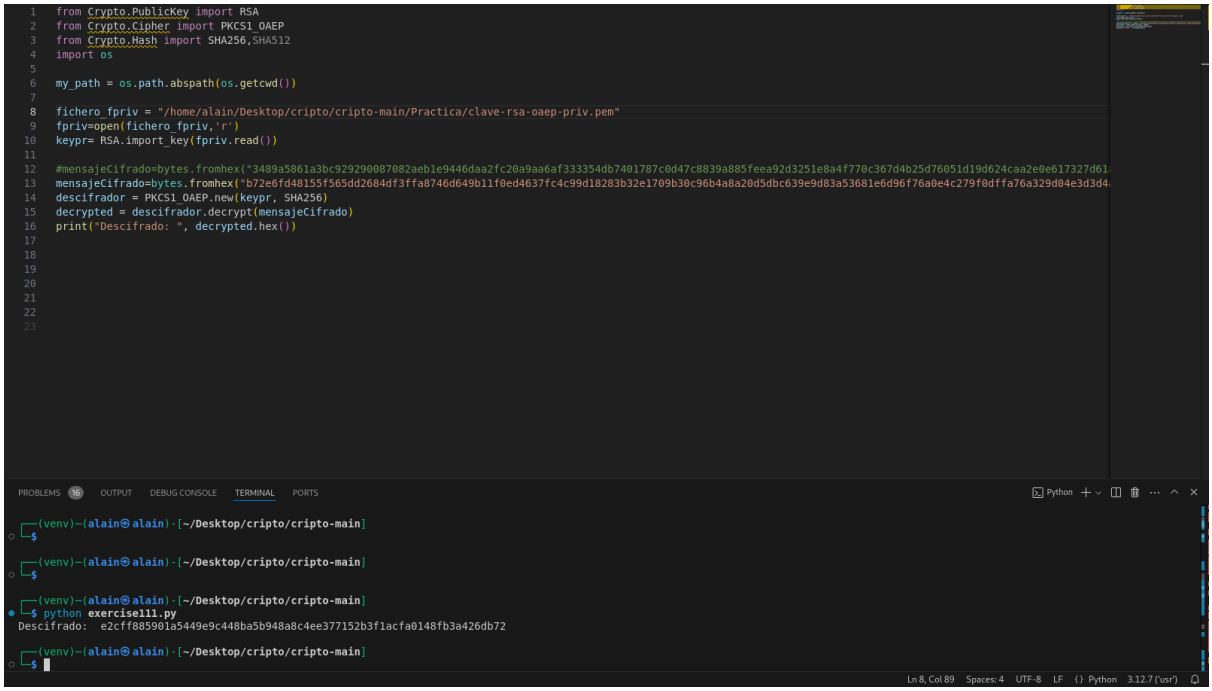
11. Nuestra compañía tiene un contrato con una empresa que nos da un servicio de almacenamiento de información de videollamadas. Para lo cual, la misma nos envía la clave simétrica de cada videollamada cifrada usando un RSA-OAEP. El hash que usa el algoritmo interno es un SHA-256.

El texto cifrado es el siguiente:

b72e6fd48155f565dd2684df3ffa8746d649b11f0ed4637fc4c99d18283b32e1709b30c
96b4a8a20d5dbc639e9d83a53681e6d96f76a0e4c279f0dffa76a329d04e3d3d4ad629
793eb00cc76d10fc00475eb76bfbc1273303882609957c4c0ae2c4f5ba670a4126f2f14
a9f4b6f41aa2edba01b4bd586624659fca82f5b4970186502de8624071be78ccef573d
896b8eac86f5d43ca7b10b59be4acf8f8e0498a455da04f67d3f98b4cd907f27639f4b1
df3c50e05d5bf63768088226e2a9177485c54f72407fdf358fe64479677d8296ad38c6f
177ea7cb74927651cf24b01dee27895d4f05fb5c161957845cd1b5848ed64ed3b0372

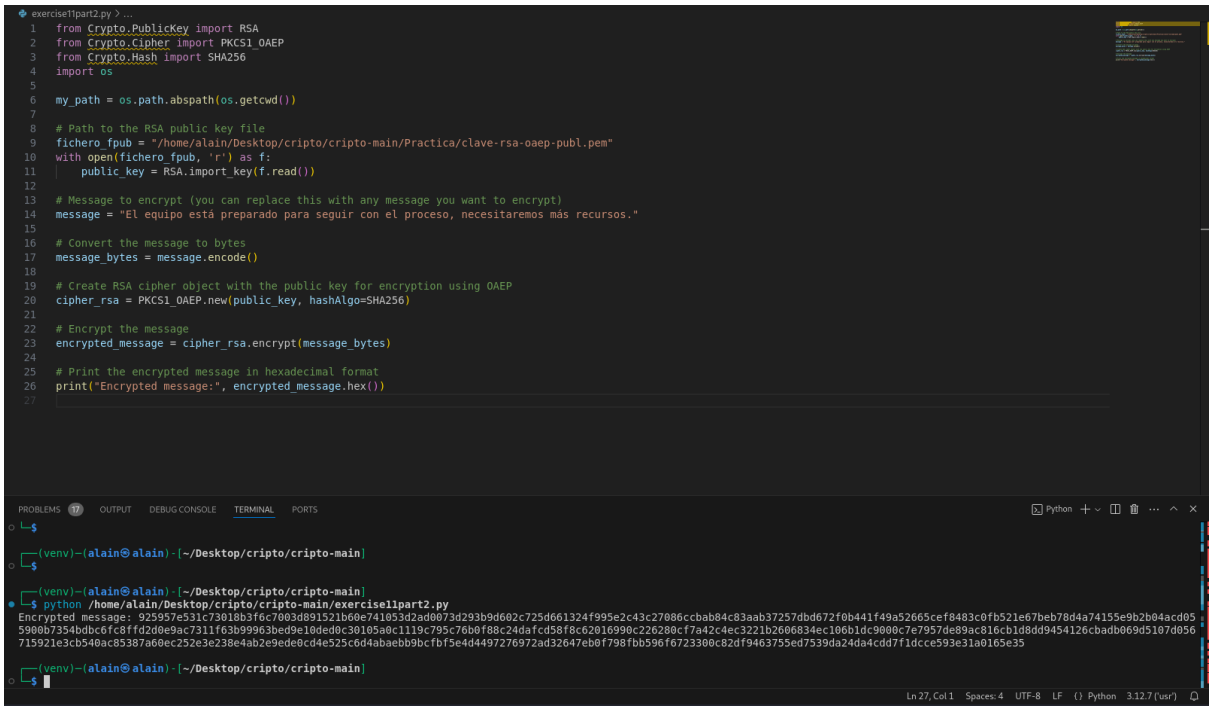
2b21a526a6e447cb8ee

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



The result:
e2cff885901a5449e9c448ba5b948a8c4ee377152b3f1acfa0148fb3a426db72

Si has recuperado la clave, vuelve a cifrarla con el mismo algoritmo.



The result:
925957e531c73018b3f6c7003d891521b60e741053d2ad0073d293b9d602c725d661324f995

e2c43c27086ccbab84c83aab37257dbd672f0b441f49a52665cef8483c0fb521e67beb78d4a7
4155e9b2b04acd055900b7354bdbc6fc8ffd2d0e9ac7311f63b99963bed9e10ded0c30105a0c
1119c795c76b0f88c24dafcd58f8c62016990c226280cf7a42c4ec3221b2606834ec106b1dc90
00c7e7957de89ac816cb1d8dd9454126cbadb069d5107d056715921e3cb540ac85387a60ec2
52e3e238e4ab2e9ede0cd4e525c6d4abaebb9bcfbf5e4d4497276972ad32647eb0f798fbb59
6f6723300c82df9463755ed7539da24da4cdd7f1dcce593e31a0165e35

¿Por qué son diferentes los textos cifrados?

The reason the two ciphered texts are different, is due to the algorithm being random by nature. RSA-OAEP involves randomness in its encryption process. When the message is encrypted with RSA-OAEP, the algorithm generates random padding that changes each time it is used, even if the input message is the same.

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:

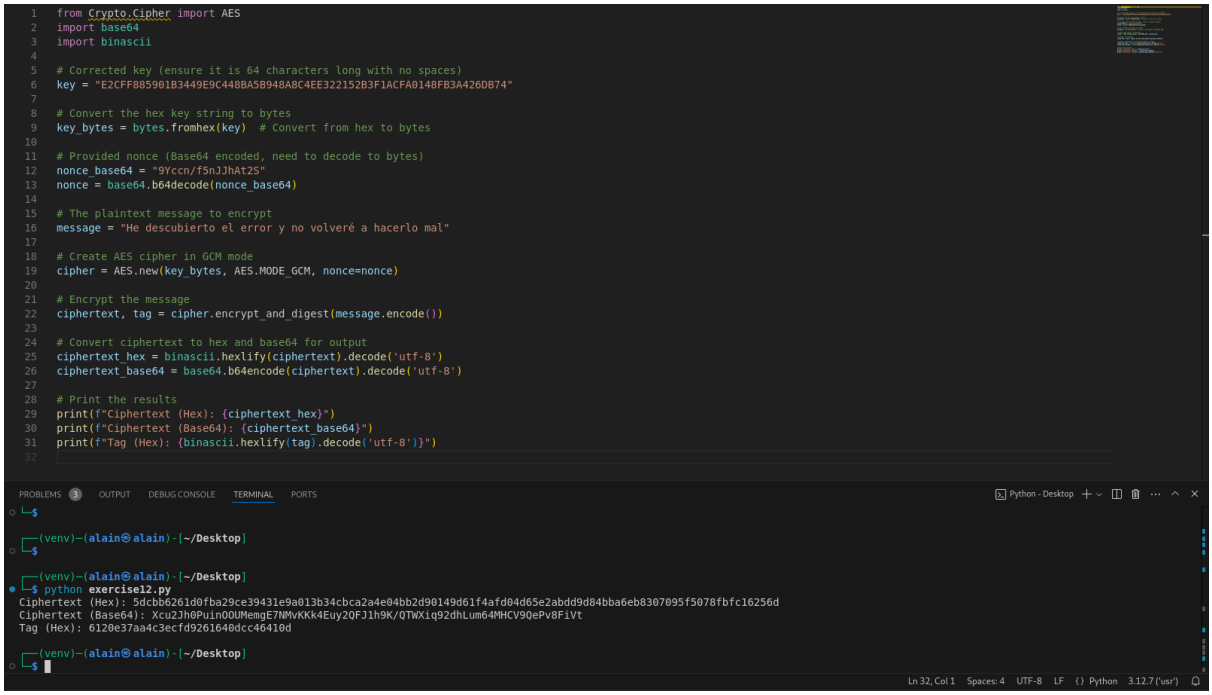
Key:E2CFF885901B3449E9C448BA5B948A8C4EE322152B3F1ACFA0148FB3A42

6DB74

Nonce:9Yccn/f5nJJhAt2S

¿Qué estamos haciendo mal?

The nonce is incorrect. It's already encoded into base64, when it should be in hexadecimal. After converting the nonce to the correct value, you should get the following:



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. **(La respuesta está en la próxima página)**

The result:
Ciphertext (Hex):
5dcbb6261d0fba29ce39431e9a013b34cbca2a4e04bb2d90149d61f4afd04d65e2abdd9d84bba6eb8307095f5078fbfc16256d
Ciphertext (Base64):
Xcu2Jh0PuinOOUMemgE7NMvKKk4Euy2QFJ1h9K/QTWXiq92dhLum64MHCV9QePv8FiVt
Tag (Hex): 6120e37aa4c3ecfd9261640dcc46410d

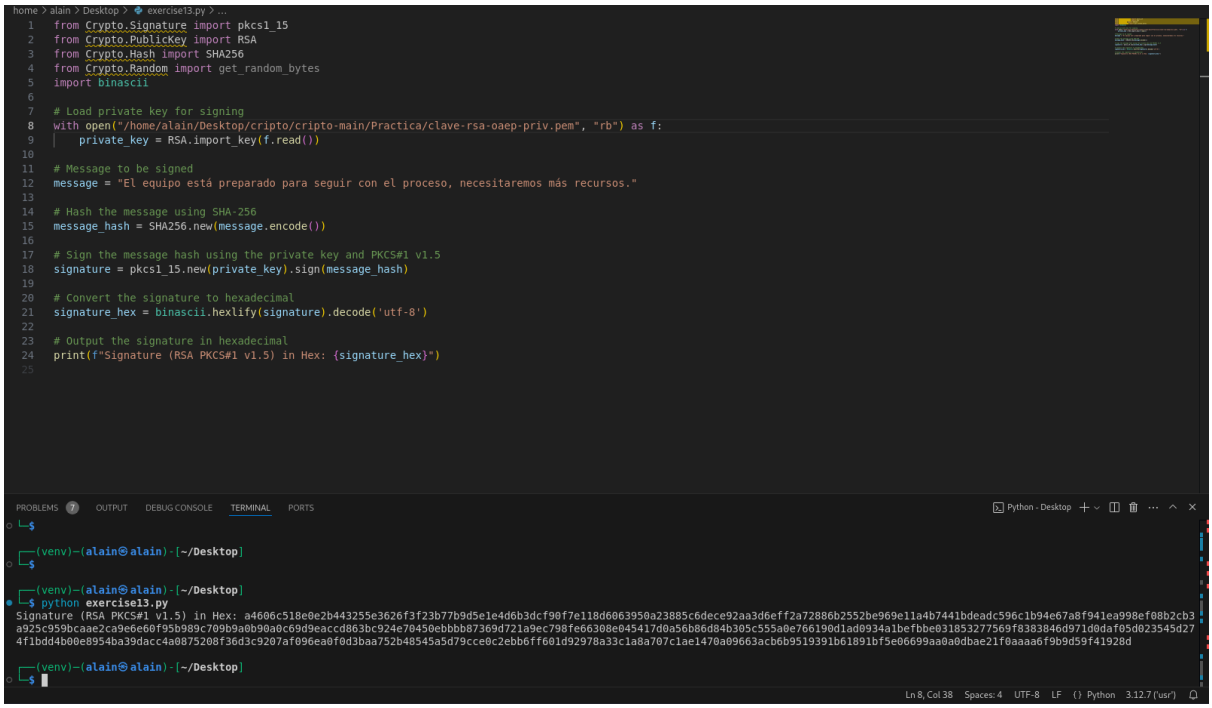
13. Se desea calcular una firma con el algoritmo PKCS#1 v1.5 usando las claves contenidas en los ficheros clave-rsa-oaep-priv y clave-rsa-oaep-publ.pem del mensaje siguiente:

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

¿Cuál es el valor de la firma en hexadecimal?

The result:

Signature (RSA PKCS#1 v1.5) in Hex:
a4606c518e0e2b443255e3626f3f23b77b9d5e1e4d6b3dcf90f7e118d6063950a23885c6dece92aa3d6eff2a72886b2552be969e11a4b7441bdeadc596c1b94e67a8f941ea998ef08b2cb3a925c959bcaae2ca9e6e60f95b989c709b9a0b90a0c69d9eaccd863bc924e70450ebbbb87369d721a9ec798fe66308e045417d0a56b86d84b305c555a0e766190d1ad0934a1befbbe031853277569f8383846d971d0daf05d023545d274f1bdd4b00e8954ba39dacc4a0875208f36d3c9207af096ea0f0d3baa752b48545a5d79cce0c2ebb6ff601d92978a33c1a8a707c1ae1470a09663acb6b9519391b61891bf5e06699aa0a0dbae21f0aaaa6f9b9d59f41928d



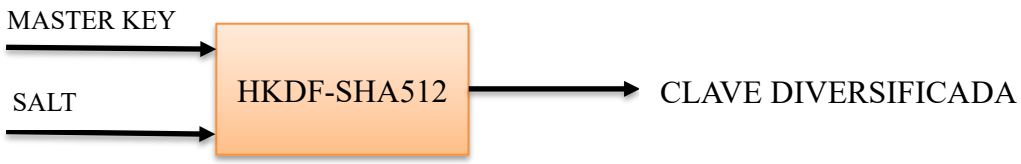
Calcula la firma (en hexadecimal) con la curva elíptica ed25519, usando las claves ed25519priv y ed25519-publ.

My kali-linux is incapable of downloading ED25519 without having a massively difficult error to resolve, but the code in the following screenshot should hypothetically be the correct code to calculate the signature using ED25519.

```python
from ecdsa import SigningKey, NIST384p
import binascii

# Load the private key for Ed25519
with open("/home/alain/Desktop/cripto/cripto-main/Practica/ed25519-priv", "rb") as f:
    private_key_ed25519 = f.read()

# Create the SigningKey object using the private key
signing_key = SigningKey.from_string(private_key_ed25519, curve=NIST384p)

# Message to be signed
message = "El equipo está preparado para seguir con el proceso, necesitaremos más recursos."

# Sign the message
signature_ed25519 = signing_key.sign(message.encode())

# Convert the signature to hexadecimal
signature_ed25519_hex = binascii.hexlify(signature_ed25519).decode('utf-8')

# Output the Ed25519 signature in hexadecimal
print(f"Signature (Ed25519) in Hex: {signature_ed25519_hex}")
```

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?

```python
from cryptography.hazmat.primitives.kdf.hkdf import HKDF
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.backends import default_backend

# Actual master key (replace this with the actual key from your keystore in hexadecimal form)
master_key = bytes.fromhex('1234567890abcdef1234567890abcdef')  # Replace with your actual key

# Device identifier
device_id = bytes.fromhex('e43bb4067cbcfab3bec54437b84bef4623e345682d89de9948fbb0afedc461a3')

# Initialize HKDF with SHA-512
hkdf = HKDF(
    algorithm=hashes.SHA512(),  # Using SHA-512 as the hash function
    length=32,  # Length of the derived key (32 bytes for AES-256)
    salt=None,  # No salt provided
    info=device_id,  # Device identifier as context (info)
    backend=default_backend()
)

# Derive the key
derived_key = hkdf.derive(master_key)
print(derived_key.hex())  # Print the derived key in hexadecimal format
```

```
PROBLEMS 7  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                Python - Desktop

┌──(venv)─(alain㉿alain)-[~/Desktop]
└─$

┌──(venv)─(alain㉿alain)-[~/Desktop]
└─$

┌──(venv)─(alain㉿alain)-[~/Desktop]
└─$ python exercise14.py
024500992de30439217e9025f5f500cd87930bd0017e98ac0efbed5eb535848b

┌──(venv)─(alain㉿alain)-[~/Desktop]
└─$

Ln 23, Col 1  Spaces: 4  UTF-8  LF  {} Python  3.12.7 ('usr')
```

The result:
024500992de30439217e9025f5f500cd87930bd0017e98ac0efbed5eb535848b

15. Nos envían un bloque TR31:

D0144D0AB00S000042766B9265B2DF93AE6E29B58135B77A2F616C8D515ACDBE6A5626F7
9FA7B4071E9EE1423C6D7970FA2B965D18B23922B5B2E5657495E03CD857FD37018E111B

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

A1A10101010101010101010101010102

¿Con qué algoritmo se ha protegido el bloque de clave?
Based on the structure of the TR31 block and the format of the transport key, the algorithm is most likely 3DES.

¿Para qué algoritmo se ha definido la clave?
The algorithm in which the key is defined for is AES.

¿Para qué modo de uso se ha generado?
The mode of use it's been generated for is encryption/decryption in CBC or ECB.

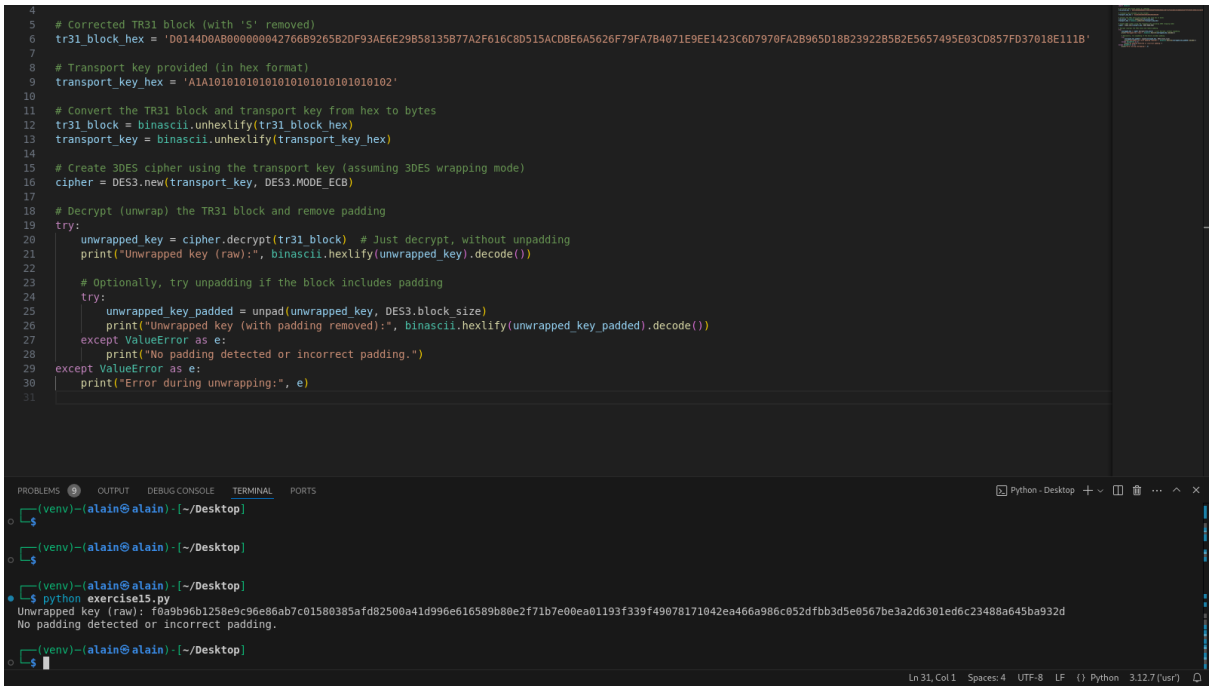
¿Es exportable?

Most likely yes.

¿Para qué se puede usar la clave?

It can be used for virtually anything related to security; such as symmetric encryption, data confidentiality, message integrity/authentication, digital signatures, or key exchange/key derivation.

¿Qué valor tiene la clave?



The result:

f0a9b96b1258e9c96e86ab7c01580385afd82500a41d996e616589b80e2f71b7e00ea01193f339f49078171042ea466a986c052dfbb3d5e0567be3a2d6301ed6c23488a645ba932d