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?

   8653f75d31455c0

   Se adjunta solución en archivo Ejercicio_1.py

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?

   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?

   En este caso funciona el descifrado en los 2 modos al ser un padding válido para ambos

   ¿Cuánto padding se ha añadido en el cifrado?

   Solo un bloque, el 01 final

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

Se adjunta solución en archivo Ejercicio_2.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.

   La etiqueta es cifrado-sim-chacha20-256

   ¿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.

   La propuesta de mejora es cifrar con un algoritmo Chacha20_Poly1305, usar un nonce aleatorio en cada cifrado y por último añadir datos asociados al mensaje cifrado. Con estas 3 mejoras se garantiza confidencialidad e integridad.

   Se adjunta solución en archivo Ejercicio_3.py

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

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmlvIjoiRG9uIFBlcGl0byBkZSB
sb3MgcGFsb3RlcyIsInJvbCI6ImlzTm9ybWFsIiwiaWF0IjoxNjY3OTMzNTMzfQ.gfhw0
dDxp6oixMLXXRP97W4TDTrv0y7B5YjD0U8ixrE
```

   ¿Qué algoritmo de firma hemos realizado?

   El algoritmo HS256 (ver cyberchef y código archivo Ejercicio_4.py)

   ¿Cuál es el body del jwt?

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

   Se adjunta url de cyberchef con la conversión y el código en archivo Ejercicio_4.py (la conversión del body légitimo falla en código por lo que comentamos de base64url)

   https://gchq.github.io/CyberChef/#recipe=From_Base64('A-Za-z0-9%2B/%3D',true,false)&input=ZXlKMGVYQWlPaUpLVjFRaUxDSmhiR2NpT2lKSVV6STFOaUo5LmV5SjFjM1ZoY21sdklqb2lSRzl1SUZCbGNHbDBieUJrWlNCc2IzTWdjR0ZzYjNSbGN5SXNJbkp2YkNJNkltbHpUbTl5YldGc0lpd2lhV0YwSWpveE5qWTNPVE16TlRNemZR&ieol=CRLF&oeol=CR

Un hacker está enviando a nuestro sistema el siguiente jwt:

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmlvIjoiRG9uIFBlcGl0byBk
ZSBsb3MgcGFsb3RlcyIsInJvbCI6ImlzQWRtaW4iLCJpYXQiOjE2Njc5MzM1MzN9
.krgBkzCBQ5WZ8JnZHuRvmnAZdg4ZMeRNv2CIAODlHRI

¿Qué está intentando realizar?

Está intentando que se valide un mensaje falso con datos incorrectos para dar rol de administrador.

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

https://gchq.github.io/CyberChef/#recipe=From_Base64('A-Za-z0-9%2B/%3D',true,false)&input=ZXlKMGVYQWlPaUpLVjFRaUxDSmhiR2NpT2lKSVV6STFOaUo5LmV5SjFjM1ZoY21sdklqb2lSRzl1SUZCbGNHbDBieUJrIFpTQnNiM01nY0dGc2IzUmxjeUlzSW5KdmJDSTZJbWx6UVdSdGFXNGlMQ0pwWVhRaU9qRTJOamM1TXpNMU16Tjk&ieol=CRLF&oeol=CR

¿Qué ocurre si intentamos validarlo con pyjwt?

https://gchq.github.io/CyberChef/#recipe=JWT_Verify('Con%20KeepCoding%20aprendemos')&input=ZXlKMGVYQWlPaUpLVjFRaUxDSmhiR2NpT2lKSVV6STFOaUo5LmV5SjFjM1ZoY21sdklqb2lSRzl1SUZCbGNHbDBieUJrWlNCc2IzTWdjR0ZzYjNSbGN5SXNJbkp2YkNJNkltbHpRV1J0YVc0aUxDSnBaWFFpT2pFMk5qYzVNek0xTXpOOS5rcmdCa3pDQlE1V1o4Sm5aSHVSdm1uQVpkZzRaTWVSTnYyQ0lBT0RsSFJJ&ieol=CRLF

Dará error debido a que no se puede validar con la clave proporcionada (ver código archivo Ejercicio_4.py)

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?

Un SHA3-256 (32 bytes)

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

4cec5a9f85dcc5c4c6ccb603d124cf1cdc6dfe836459551a1044f4f2908aa5d63739506f
6468833d77c07cfd69c488823b8d858283f1d05877120e8c5351c833

¿Qué hash hemos realizado?

Un SHA512 (64 bytes)

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?

Que con solo añadir un caracter, la cadena resultante es completamente distinta, se le llama efecto avalancha

Se adjunta solución en archivo Ejercicio_5.py

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.

   HMAC:
   857d5ab916789620f35bcfe6a1a5f4ce98200180cc8549e6ec83f408e8ca0550

   Se adjunta solución en archivo Ejercicio_6.py

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?

   El SHA-1 se considera inseguro desde 2005 y desde 2017 se han encontrado ataques de colisión

   Después de meditarlo, propone almacenarlo con un SHA-256, y su responsable le pregunta si no lo va a fortalecer de alguna forma. ¿Qué se te ocurre?

   Fortalecerlo mediante SALT, al concatenar un dato contenido en la base de datos, por ejemplo el nombre de usuario (aunque sería preferible usar un valor aleatorio), con el password, se evita que se puedan usar rainbow tables de hashes precalculados. Se puede fortalecer más con PEPPER, el concepto es el mismo que el SALT, pero el dato a concatenar no se encuentra contenido en la base de datos, (es secreto y se tiene que almacenar aparte) aumentando así la seguridad al añadir aún más entropía. Normalmente es un valor único por cada base de datos.

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

Implementar Argon2id que ralentiza notablemente los ataques y es específico para passwords

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?

Usaría AES-GCM ya que garantiza tanto confidencialidad como integridad

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.

   KCV AES:
   5244dbd02d57d56ae08e064c56c7ca74a35eccad6db31f05841bde3d4e3ada4a
   KCV SHA256: db7df2

   Se adjunta solución en archivo Ejercicio_9.py

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.

Se adjuntan archivos y capturas de pantalla en la carpeta correspondiente al ejercicio 10

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

    177ea7cb74927651cf24b01dee27895d4f05fb5c161957845cd1b5848ed64ed3b03722b21a526a6e447cb8ee

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

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

    clave
    e2cff885901a5449e9c448ba5b948a8c4ee377152b3f1acfa0148fb3a426db72

    Cifrado:
    24a1423b5685a94ed708b029dd95b0acdb3fb3e16a9a9de1c29e3dc4491636c73f40c7038d033344dfa334a2c66fb1000c9f2f284b1f576292ab774ed87bda9

43f96aed431d9750e002232acc426ed3172b52170c5fb6dadf6fbba0d4448bd52bbd7116a8a1f36b9d24ff8b6d0d09b9913dffc806edb6b2329b256d4a37a1e4fb7ad69da8b3faf259832b6c3e37341f7a24b52012889cfcf0d159cfd485c15758bd099882945e88c570ea7b5faf616c1790c7864fe76525d4353204083cbf36965879b5b71f62aaf3af7ad6532d8e38d691dd298aaf77191e9102c99167d6ccb9d67b9ea62b60e1fa1a48130ed275601a9c47d6ba3e4b8ab76f0599252577d25

Porque el RSA-OAEP añade aleatoriedad en el cifrado mediante padding, de ahí que no puedan coincidir ambos cifrados.

Se adjunta solución en archivo Ejercicio_11.py

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?

El Nonce tiene que ser siempre aleatorio para que el AES/GCM sea seguro.

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 preséntalo en hexadecimal y en base64.

El texto en hexadecimal es:
5dcbb6261d0fba29ce39431e9a013b34cbca2a4e04bb2d90149d61f4afd04d65e2abdd9d84bba6eb8307095f5078fbfc16256d

El texto en Base64 es:
Xcu2Jh0PuinOOUMemgE7NMvKKk4Euy2QFJ1h9K/QTWXiq92dhLum64MHCV9QePv8FiVt

El tag en hexadecimal es: 6120e37aa4c3ecfd9261640dcc46410d
El tag en Base64 es: YSDjeqTD7P2SYWQNzEZBDQ==

Se adjunta solución en archivo Ejercicio_12.py

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?

Firma:
a4606c518e0e2b443255e3626f3f23b77b9d5e1e4d6b3dcf90f7e118d6063950a23885c6dece92aa3d6eff2a72886b2552be969e11a4b7441bdeadc596c1b94e67a8f941ea998ef08b2cb3a925c959bcaae2ca9e6e60f95b989c709b9a0b90a0c69d9eaccd863bc924e70450ebbbb87369d721a9ec798fe66308e045417d0a56b86d84b305c555a0e766190d1ad0934a1befbbe031853277569f8383846d971d0daf05d023545d274f1bdd4b00e8954ba39dacc4a0875208f36d3c9207af096ea0f0d3baa752b48545a5d79cce0c2ebb6ff601d92978a33c1a8a707c1ae1470a09663acb6b9519391b61891bf5e06699aa0a0dbae21f0aaaa6f9b9d59f41928d

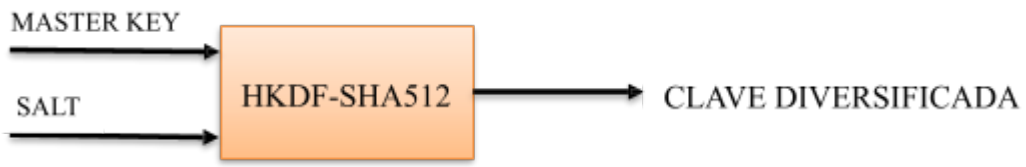
Calcula la firma (en hexadecimal) con la curva elíptica ed25519, usando las claves ed25519- priv y ed25519-publ.

b'bf32592dc235a26e31e231063a1984bb75ffd9dc5550cf30105911ca4560dab52abb40e4f7e2d3af828abac1467d95d668a80395e0a71c51798bd54469b7360d'

Se adjunta solución en archivo Ejercicio_13.py

14. Necesitamos generar una nueva clave AES, usando para ello una HKDF (HMAC-based Extract- and-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?

Clave diversificada:
e716754c67614c53bd9bab176022c952a08e56f07744d6c9edb8c934f52e448a

Se adjunta solución en archivo Ejercicio_14.py

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? **La letra D indica que es AES**
¿Para qué algoritmo se ha definido la clave? **Letra A indica que es AES**
¿Para qué modo de uso se ha generado? **La letra B indica cifrar y descifrar**
¿Es exportable? **Letra S indica que es exportable bajo clave no confiable**
¿Para qué se puede usar la clave? **La clave D0 significa para cifrado simétrico**
¿Qué valor tiene la clave? **c1c1c1c1c1c1c1c1c1c1c1c1c1c1c1c1**

Se adjunta solución en archivo Ejercicio_15.py