KeepCoding Bootcamp Ciberseguridad | Edición IX

Módulo de Criptografía

Informe Práctica Criptografía

Por: Oscar Uriel Tobar Rios
Fecha del Informe: 21/12/2024

Contenido

1 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.

a. 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?

   SOLUCION

   Si A es clave fija , B es la clave del archivo de propiedades (Key Manager) y C es la clave final en memoria, podríamos deducir la clave según las propiedades asociativas

   $(A \oplus B) = C$

   $(B \oplus C) = A$

   $(A \oplus C) = B$

   Clave fija en código = B1EF2ACFE2BAEEFF

   Clave final (en memoria) es 91BA13BA21AABB12

   Para scar el valor del key manager lo hacemos asi

   $(A \oplus C) = B$

   B1EF2ACFE2BAEEFF $\oplus$ 91BA13BA21AABB12 = 20553975C31055ED

   RTA: el valor en el key Manager es 20553975C31055ED

   Solución en Python en

   https://github.com/oscartobar/practicaskeepcoding/blob/main/Criptografia/practica/practica_final_punto1.py

b. 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?

   SOLUCION

   Si A es clave fija , B es la clave del archivo de propiedades (Key Manager) la clave final en memoria será C

   $(A \oplus B) = C$

   Clave fija en código = B1EF2ACFE2BAEEFF

   Clave del KeyManager es B98A15BA31AEBB3F

Para sacar el valor de la clave en memoria lo hacemos asi
(A ⊕B)=C
B1EF2ACFE2BAEEFF ⊕ 91BA13BA21AABB12 = 8653F75D31455C0

RTA: el valor de la clave en memoria es 08653F75D31455C0
Solución en Python en
https://github.com/oscartobar/practicaskeepcoding/blob/main/Criptografia/practica/practica_final_punto1b.py

2 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/o0QKIWXg3nu1RRz4QWElezdrLAD5LO4USt3aB/i50nvvJbBiG+le1ZhpR84oI=

a. Para este caso, se ha usado un AES/CBC/PKCS7. Si lo desciframos, ¿qué obtenemos?
RESPUESTA: Al descifrar el dato obtenemos la siguiente cadena “**Esto es un cifrado en bloque típico. Recuerda, vas por el buen camino. Ánimo.”**
Solución en Python en
https://github.com/oscartobar/practicaskeepcoding/blob/main/Criptografia/practica/practica_final_punto2.py

b. ¿Qué ocurre si decidimos cambiar el padding a x923 en el descifrado?
RESPUESTA: es mensaje se corrompe Al descifrar el dato obtenemos la siguiente cadena: ❒Tj{cR❒❒❒y❒❒→❒❒do en bloque típico. Recuerda, vas por el buen camino.
Ánimo.**https://github.com/oscartobar/practicaskeepcoding/blob/main/Criptografia/practica/practica_final_punto2b.py**

¿Cuánto padding se ha añadido en el cifrado?
RESPUESTA: Añade un carácter de padding. Respuesta en Python obteniendo el dato de la clave desde el keystore

SOLUCION EN PYTHON:

https://github.com/oscartobar/practicaskeepcoding/blob/main/Criptografia/practica/practica_final_punto2b.py

3 Ejercicio 3

a) 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.

SOLUCION

La clave es: af9df30474898787a45605ccb9b936d33b780d03cabc81719d52383480dc3120

Nonce: WkUWyUg/0CkrkNHE

Texto Encriptado: c1gc10262NrJbupSISQwo3AAkrYeLg9Rkx/pT9ixZZYSzbnZxTvHLZpCk10=

Solución en Python

https://github.com/oscartobar/practicaskeepcoding/blob/main/Criptografia/practica/practica_final_punto3.py

b) ¿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.

SOLUCION

Podemos mejorar el sistema para garantizar la integridad del mensaje utilizando el algoritmo Chacha20-Poly1305 utilizando datos_asociados para garantizarlo

Solución en Python

https://github.com/oscartobar/practicaskeepcoding/blob/main/Criptografia/practica/practica_final_punto3b.py

4 Ejercicio 4

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

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmlvIjoiRG9uIFBlcGl0byBkZSBsb3MgcGFsb3RlcyIsInJvbCI6ImlzTm9ybWFsIiwiaWF0IjoxNjY3OTMzNTMzfQ.gfhw0dDxp6oixMLXXRP97W4TDTrv0y7B5YjD0U8ixrE

¿Qué algoritmo de firma hemos realizado?

¿Cuál es el body del jwt?

SOLUCION

El algoritmo de firma es HS256

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

El Body es

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

"iat": 1667933533
}

Solución en Python
https://github.com/oscartobar/practicaskeepcoding/blob/main/Criptografia/practica/practica_final_punto4.py

b. Un hacker está enviando a nuestro sistema el siguiente jwt:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmlvIjoiRG9uIFBlcGl0byBkZSBsb3MgcGFsb3RlcyIsInJvbCI6ImlzQWRtaW4iLCJpYXQiOjE2Njc5MzM1MzN9.krgBkzCBQ5WZ8JnZHuRvmnAZdg4ZMeRNv2ClAODlHRI
¿Qué está intentando realizar?
¿Qué ocurre si intentamos validarlo con pyjwt?
SOLUCION:
Se está intentando realizar un envío de un token que invalido pues su firma no es la esperada
Lo que ocurre si intentamos validarlo con pyjwt es que sale un error "Signature verification failed"
Solución en Python
https://github.com/oscartobar/practicaskeepcoding/blob/main/Criptografia/practica/practica_final_punto4b.py

5 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?
SOLUCION: Por la longitud es de sha3_256

SOLUCION EN PYTHON:

https://github.com/oscartobar/practicaskeepcoding/blob/main/Criptografia/practica/practica_final_punto5.py

Y si hacemos un SHA2, y obtenemos el siguiente resultado:
4cec5a9f85dcc5c4c6ccb603d124cf1cdc6dfe836459551a1044f4f2908aa5d63739506f6468833d77c07cfd69c488823b8d858283f1d05877120e8c5351c833

¿Qué hash hemos realizado?
SOLUCION: se ha realizado un hash sha512
SOLUCION EN PYTHON:

https://github.com/oscartobar/practicaskeepcoding/blob/main/Criptografia/practica/practica_final_punto5b.py

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?

SOLUCION: El hash es único para cada cadena y cada tipo de hash

SOLUCION EN PYTHON:

https://github.com/oscartobar/practicaskeepcoding/blob/main/Criptografia/practica/practica_final_punto5c.py

6 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.

SOLUCION:
857d5ab916789620f35bcfe6a1a5f4ce98200180cc8549e6ec83f408e8ca0550

SOLUCION EN PYTHON:

https://github.com/oscartobar/practicaskeepcoding/blob/main/Criptografia/practica/practica_final_punto6.py

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

SOLUCION:

¿Por qué crees que es una mala opción SHA-1? RTA: Es un algoritmo de Hash no recomendado debido a que se considera inseguro por ser vulnerable a ataques de colisión

Como fortalecer de alguna forma SHA-256. ¿Qué se te ocurre? RTA se puede usar un SALT (o un valor aleatorio) para ayudar a prevenir ataques de diccionario, y se puede además un secreto global (PEPPER) para fortalecer el SALT. Finalmente se puede garantizar la autenticidad del mensaje e integridad utilizando una clave secreta (HMAC), para evitar ataques de manipulación de datos.

hay margen de mejora de SHA-256. ¿Qué propondrías? Claro en este caso lo mas recomendado seria usar un algoritmo especializado para proteger contraseñas, como es el ARGON2, que es un algoritmo moderno para este propósito.

8 Ejercicio 8

Tenemos la siguiente API REST, muy simple. Request:

{"idUsuario":1,"usuario":"José Manuel Barrio Barrio","tarjeta":4231212345676891}

Response:

{ "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?
SOLUCION

Lo que se debe realizar en cifrar el cuerpo del request de tal manera que la data enviada no sea entendible en el transporte del mensaje es decir que viaje cifrada la data.

Para ello podemos asegurar la confidencialidad y la integridad del mensaje usando el algoritmo AES-256. Para cifrarlo utilizamos la función AES_Encrypt y le enviamos el mensaje (en JSON), un IV y una llave compartida. Para garantizar la integridad podemos usar un HMAC con otra clave compartida para garantizar que el mensaje envido no sea modificado en transito.

En el destino para poder leer el mensaje debemos enviarles el IV, el HMAC y en el cuerpo del mensaje en la data el mensaje JSON original, cifrado con AES-256 y codificado en Base64.

Del lado del servidor destino tendrán que extraer el IV y el cuerpo cifrado, calcula el HMAC con la misma clave compartida y verificar que coincida y finalmente descifrar el cuerpo usando el IV y la clave AES compartida.

9 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.

SOLUCION:

KCV SHA256: DB7DF2

KCV AES: 5244DBD02D57

SOLUCION EN PYTHON:

https://github.com/oscartobar/practicaskeepcoding/blob/main/Criptografia/practica/practica_final_punto9.py

10 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.

```
pub   ed25519 2022-06-26 [SC]
      1BDE635E4EAE6E68DFAD2F7CD730BE196E466101
uid         [desconocida] Pedro Pedrito Pedro <pedro.pedrito.pedro@empresa.com>
sub   cv25519 2022-06-26 [E]
```

```
pub   ed25519 2022-06-26 [SC]
      F2B1D0E8958DF2D3BDB6A1053869803C684D287B
uid         [desconocida] RRHH <RRHH@RRHH>
sub   cv25519 2022-06-26 [E]
```

```
C:\Users\ecm8582a>gpg --list-secret-keys
[keyboxd]
---------
sec   ed25519 2022-06-26 [SC]
      1BDE635E4EAE6E68DFAD2F7CD730BE196E466101
uid         [desconocida] Pedro Pedrito Pedro <pedro.pedrito.pedro@empresa.com>
ssb   cv25519 2022-06-26 [E]
```

```
sec   ed25519 2022-06-26 [SC]
      F2B1D0E8958DF2D3BDB6A1053869803C684D287B
uid         [desconocida] RRHH <RRHH@RRHH>
ssb   cv25519 2022-06-26 [E]
```

Se requiere verificar la misma, y evidenciar dicha prueba.

```
C:\Users\ecm8582a>gpg --output xxx.txt --decrypt D:\repositorios\keepcoding\github\cripto\Practica\MensajeRespoDeRaulARRHH.sig
gpg: Firmado el 06/26/22 06:47:01 Hora est. Pacfico, Sudamrica
gpg:                usando EDDSA clave 1BDE635E4EAE6E68DFAD2F7CD730BE196E466101
gpg:                emisor "pedro.pedrito.pedro@empresa.com"
gpg: Firma correcta de "Pedro Pedrito Pedro <pedro.pedrito.pedro@empresa.com>" [desconocido]
gpg: WARNING: The key's User ID is not certified with a trusted signature!
gpg:          No hay indicios de que la firma pertenezca al propietario.
Huellas dactilares de la clave primaria: 1BDE 635E 4EAE 6E68 DFAD  2F7C D730 BE19 6E46 6101
```

```
C:\Users\ecm8582a>more xxx.txt
Se debe ascender inmediatamente a Raúl. Es necesario mejorarle sus condiciones económicas un 20% para que se quede con nosotros.
```

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.

```
C:\Users\ecm8582a>more mensaje.txt.asc
-----BEGIN PGP MESSAGE-----

owGbwMvMwCVmkdlgk+GrUc24piiJOzc1rzgxK1WvpKIkPe0da1hmal5KvkJxqUJB
alFaZo5Cap5Cao5CbmpRcmJKvo5CRmpufrFCSmpyZkomUF1icTJQfWpRTqpCJVBR
Vn5RIohdmqdgZKoKMqU4MSexKDNfTyE4Mac0Jb9Yr6OUhUGMi0FWTJHl08YLL6b2
frq8d9tCVpibWJlArmDg4hSAiYiEMfzTtb65/Eu9M1cI/6/na7ZevtVb9itY+sWe
8FXRVw6u+HvrHSPD9v3RLh4++idnXV8tfXAVQ2Uz+5+0Rv2ne3vZ9Hi1E5p5AQ==
=gx95
-----END PGP MESSAGE-----
```

https://github.com/oscartobar/practicaskeepcoding/blob/main/Criptografia/practica/mensaje.txt.asc

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.

```
C:\Users\ecm8582a>more mensajefinal_punto10.txt.asc
-----BEGIN PGP MESSAGE-----

owGbwMvMwCV23WCfZJ5bIiPDBDDXIrPBJsNXo5pxTXmSRG5qXnFiVmpaZl5iTnxB
aV5JvqGBXklFSXra+2zX4pLE3PxihZL8FCCZkqqQmFyaWpSSr6OQmqOQWJwM1Jqv
UJxadHghSCA3tVihsDRVoSwzNS9VRyExPb+4BKi0OFMhL18hI7FSoTi/qKAotTix
WK+jlIVBjItBVkyR5dPGCy+m9n66vHfbQlaYw1iZQLYzcHEKwEQifBn+F2qvmXnr
K1tu+uk/teFMf5WvCcfe1T9V5RD8P2HltdfuvIwMLe68Nj6N6dvnz0u7deFD45mT
V5u1W4stz5Udn/SId50XJ5K10veS4/zW5WXcX6tfAwsehLUwEctOhv/OS2an+81f
m6ps99l6wUzJFPXeS6a33LmC5n+pdl54Lec/I8P39SZ3q7Iv/CjZZFG4vbPD8kbd
zHtGV98e/MxsOcVoRhAjAA==
=rwdF
-----END PGP MESSAGE-----
```

https://github.com/oscartobar/practicaskeepcoding/blob/main/Criptografia/practica/mensajefinal_punto10.txt.asc

SOLICIONES:

Los comandos ejecutados se encuentran en

https://github.com/oscartobar/practicaskeepcoding/blob/main/Criptografia/practica/practica_final_punto10.txt

11 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
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. Si has recuperado la clave, vuelve a cifrarla con el mismo algoritmo. ¿Por qué son diferentes los textos cifrados?

SOLUCION

Descifrado: e2cff885901a5449e9c448ba5b948a8c4ee377152b3f1acfa0148fb3a426db72

Cifrado:
43abf1c435e687391cfe6de0ea988baf2294b5c5dda69200180305335c561d8913b9fc4dfeda9
b26f805b7109aff14669bad62e372f992139dcc013de381c06a57e788709c9c57c231dac11ac
c3da70ecdd4562e155dcc2e08e3fd3cbdaf25990cb2ed242d2143cf8b2c0c7539a850f356450
8e6994d05146faf77fbef0c3555df2df859d076df826c6932bff640519c6e01a463f405c020fc29
2531d96ef6c506fccdabc456b64e3dd5c7f08f1ab4a3e3365cd1f3373397dc042461f61a2bd92
4585b6b43c567ed660fa51b853c717b8a02bbc75c06013d616612083f37a4fcfc4ab206e8245
86a8f1da57c1cef5c61bdb386a19e702e38a2dbdedfe1622200

El motivo que los textos cifrados sean diferentes incluso cuando usando las mismas claves (pública y privada) y el mismo mensaje, es porque estamos usando el algoritmo de cifrado RSA, el cual genera un texto cifrado diferente cada vez. Esto ocurre debido a la implementación de técnicas de aleatoriedad para mejorar la seguridad.

Solución en Python en
https://github.com/oscartobar/practicaskeepcoding/blob/main/Criptografia/practica/practica_final_punto11.py

12Ejercicio 12

a. 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?

SOLUCION
El Nonce en AES/GCM debe ser único y no repetirse jamás con la misma clave. Si se reutiliza, se compromete la seguridad de la comunicación. Los atacantes podrían deducir información sensible del mensaje cifrado al observar patrones o diferencias en los datos cifrados con el mismo Nonce. Solución en Python con Nonce Dinámico

SOLUCION EN PYTHON:

https://github.com/oscartobar/practicaskeepcoding/blob/main/Criptografia/practica/practica_final_punto12.py

b. 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.
SOLUCION
El texto en Haxadecimal y Base64:
534755675a47567a59335669615756796447386 75a5777675a584a796233496765534275627942326232783 25a584c447153426849476868593256796247386762574673.
Solución en Python en
https://github.com/oscartobar/practicaskeepcoding/blob/main/Criptografia/practica/practica_final_punto12b.py

13Ejercicio 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

SOLUCION EN PYTHON:

https://github.com/oscartobar/practicaskeepcoding/blob/main/Criptografia/practica/practica_final_punto13.py

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

Firma Generada (64 bytes):
b'bf32592dc235a26e31e231063a1984bb75ffd9dc5550cf30105911ca4560dab52abb40e4f7e2d3af828abac1467d95d668a80395e0a71c51798bd54469b7360d'

La firma es válida

SOLUCION EN PYTHON:

https://github.com/oscartobar/practicaskeepcoding/blob/main/Criptografia/practica/practica_final_punto13b.py

14Ejercicio 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 es: e43bb4067cbcfab3bec54437b84bef4623e345682d89de9948fbb0afedc461a3

Clave key1: 4086d65b159d8b82d87eeb41978ec466945ccaffac834eea3204c2c7452127cb

Clave key2: bb9cf18496fda122bdf670554872a4669db975394d74a04019bd7de6151520c1

SOLUCION EN PYTHON:

https://github.com/oscartobar/practicaskeepcoding/blob/main/Criptografia/practica/practica_final_punto14.py

15Ejercicio 15

Nos envían un bloque TR31:

D0144D0AB00S000042766B9265B2DF93AE6E29B58135B77A2F616C8D515ACDBE6A5626F79FA7B4071E9EE1423C6D7970FA2B965D18B23922B5B2E5657495E03CD857FD37018E111B

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

A1A10101010101010101010101010102

1. ¿Con qué algoritmo se ha protegido el bloque de clave? RTA: DES
2. ¿Para qué algoritmo se ha definido la clave? RTA: AES
3. ¿Para qué modo de uso se ha generado? RTA: Both Encrypt & Decrypt / Wrap & Unwrap
4. ¿Es exportable? RTA: Sensitive, exportable under untrusted key
5. ¿Para qué se puede usar la clave? RTA: Data Encryption Key for Decimalization Table
6. ¿Qué valor tiene la clave? RTA: a2a1c1c1c1c1c1c1c1c1c1c1c1c1c1c2

SOLUCION EN PYTHON:

https://github.com/oscartobar/practicaskeepcoding/blob/main/Criptografia/practica/practica_final_punto15.py