

## 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. 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 operación a realizar es un XOR entre la clave fija y la clave final para obtener la clave en properties. El valor que ha puesto el Key Manager es: 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?

**Se realiza un XOR entre las claves fija y dinámica. La clave con la que se trabaja en memoria es: 08653f75d31455c0**

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?

**Obtenemos el texto: "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?

**Si cifras los datos utilizando PKCS7 y luego intentas descifrarlos utilizando x923, el proceso de eliminación del padding interpretará incorrectamente los bytes de padding.**

¿Cuánto padding se ha añadido en el cifrado?

Datos con padding PKCS7:

```
"M\x0fR8\xc2\x9c\xe9\xa1R\xf5)q\x85\xf2\xbd\xc1=|QzO\t\tdet\xe5w\xf9'\xfeg,\x8e\xce\x7f\xfa4
@\xa2\x16^\r\xe7\xbbTQ\xcf\x84\x16\x12W\xb3v\xb2\xc0\x0f\x92\xce\xe1D\xad\xdd\xa0\x7f\x8
b\x9d'\xbe\xf2[\x06!\xbe\x95\xedY\x86\x94|\xe2\x82\x10\x10\x10\x10\x10\x10\x10\x10\x10\x10
\x10\x10\x10\x10\x10\x10"
```

Datos con padding x923:

```
b"M\x0fR8\xc2\x9c\xe9\xa1R\xf5)q\x85\xf2\xbd\xc1=|QzO\t\tdet\xe5w\xf9'\xfeg,\x8e\xce\x7f\xfa
4@\xa2\x16^\r\xe7\xbbTQ\xcf\x84\x16\x12W\xb3v\xb2\xc0\x0f\x92\xce\xe1D\xad\xdd\xa0\x7f\x8
b\x9d'\xbe\xf2[\x06!\xbe\x95\xedY\x86\x94|\xe2\x82\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
0\x00\x00\x00\x00\x00\x10"
```

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.

**Clave extraída:**

[af9df30474898787a45605ccb9b936d33b780d03cab81719d52383480dc3120](#)

**Mensaje cifrado en HEX =**

[69ac4ee7c4c552537a00a19bc9f70aaed7c9c8f769956a09bce6f6def6c3535f2211c9467067cf5c4a842ab](#)

**Mensaje cifrado en B64 =**

[aaxO58TFUIN6AKGbyvfwqu18nI92mVagm85vre9sNTXylRyUZwZ89cSoQqs=](#)

**Mensaje en claro =**

[KeepCoding te enseña a codificar y a cifrar](#)

¿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.

Para garantizar la integridad además de la confidencialidad, podemos combinar el cifrado con Chacha20 con una firma criptográfica. Una forma común de hacerlo es utilizando un código de autenticación de mensaje (MAC), como el HMAC o el sistema ChaCha20-Poly1305 que combina el cifrado y la autenticación en un solo paso.

**Versión mejorada Chacha20-Poly1305****texto cifrado:**

[4ec95921ca8b757e2336605c7dbab8f4d40b5b4d220e66aa978f740d3e59b0cd70e3217242991cc140bb1e1a](#)

[tag: 710cd4723da6d5ef37f23bee66285e57](#)

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

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmVlIjoRG9uIFBlcGl0byBkZSBsb3MgcGFsb3RlcylsInJvbCI6ImIzTm9ybWFsIiwiaWF0IjoxNjY3OTMzNTMzZfQ.gfhw0dDxp6oixMLXXRP97W4TDTrv0y7B5YjD0U8ixrE
```

¿Qué algoritmo de firma hemos realizado?

**El algoritmo de firma es HS256 (HMAC con SHA-256).**

¿Cuál es el body del jwt?

**El body es:**

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

Un hacker está enviando a nuestro sistema el siguiente jwt

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmVlIjoRG9uIFBlcGl0byBkZSBsb3MgcGFsb3RlcylsInJvbCI6ImIzQWRtaW4iLCJpYXQiOiJlY2Njc5MzM1MzN9.krgBkzCBQ5WZ8JnZHuRvmnAZdg4ZMeRNv2CIAODIHRl
```

¿Qué está intentando realizar?

**Decodificando el base64url de la parte del payload obtenemos:**

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

**Esto indica que un atacante está intentando elevar sus privilegios cambiando el rol del usuario a administrador.**

¿Qué ocurre si intentamos validarlo con pyjwt?

**En este caso, si el JWT manipulado es enviado, se generará un error porque la firma no coincide con la clave proporcionada y el JWT ha sido alterado.**

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?

**64 caracteres hex = 256 bits (SHA3-256)**

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

```
4cec5a9f85dcc5c4c6ccb603d124cf1cdc6dfe836459551a1044f4f2908aa5d63739506f  
6468833d77c07cfd69c488823b8d858283f1d05877120e8c5351c833
```

¿Qué hash hemos realizado?

**128 caracteres hex = 512 bits (SHA2-512)**

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?

**El hash generado es el siguiente:**

**302be507113222694d8c63f9813727a85fef61a152176ca90edf1cfb952b19bf**

**Destacaría la sensibilidad a cambios.**

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.

**El HMAC-SHA256 es:**

**857d5ab916789620f35bcfe6a1a5f4ce98200180cc8549e6ec83f408e8ca0550**

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 no es adecuado para almacenar contraseñas por varias razones:**

- **Seguridad Comprometida:** SHA-1 es vulnerable a ataques de colisión, lo que significa que dos entradas diferentes pueden producir el mismo hash. Esto debilita la seguridad general del hash.
- **Ataques de Fuerza Bruta:** SHA-1 no está diseñado para ser resistente a ataques de fuerza bruta. Los atacantes pueden usar técnicas como el uso de hardware

especializado (FPGAs, GPUs) para realizar ataques de fuerza bruta de manera eficiente.

- **Velocidad:** SHA-1 es muy rápido, lo que es una desventaja cuando se usa para hashing de contraseñas porque permite a los atacantes intentar muchas contraseñas por segundo.

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?

- **Salting:** Añadir un valor aleatorio único (salt) a cada contraseña antes de hashearla para garantizar que contraseñas idénticas no produzcan el mismo hash. Esto previene los ataques de rainbow tables.
- **Iteración:** Realizar múltiples rondas de hashing para aumentar el tiempo necesario para calcular el hash de una contraseña. Esto hace que los ataques de fuerza bruta sean más costosos en términos de tiempo.

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

Utilizar algoritmos especializados como bcrypt, scrypt o Argon2. Estos algoritmos están diseñados específicamente para el almacenamiento seguro de contraseñas e incluyen características como salting e iteración.

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?

**Haría un cifrado AES:**

- Generaría una clave simétrica AES
- Cifraría los campos confidenciales (por ejemplo, “tarjeta”) con la clave AES.

Esta clave deberá ser compartida entre el cliente y el servidor (por ejemplo, utilizando una clave RSA para cifrar la clave durante la transmisión inicial.

Para garantizar la integridad del mensaje y asegurarse de que no ha sido modificado, se puede usar una firma digital. Un algoritmo comúnmente utilizado es RSA o ECDSA (Elliptic Curve Digital Signature Algorithm). Estas firmas se generan utilizando una clave privada y se verifican con la correspondiente clave pública.

**Pasos para la firma digital:**

- Generar un hash del mensaje completo (o de los campos importantes) usando un algoritmo de hash seguro como SHA-256.
- Firmar el hash con una clave privada (RSA o ECDSA).
- Incluir la firma en el mensaje enviado.

**Redefinición de la API**

Redefiniría la API para incluir el cifrado y la firma digital en el proceso de comunicación.

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.

**KCV AES:** 5244db

**KCV SHA256:** db7df2

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.

```
C:\Users\dmore\Desktop\Ciberseguridad\Criptografia\Práctica\Ejercicio 10>gpg --verify MensajeRespoDeRaulARRHH.sig
gpg: Firmado el 06/26/22 13:47:01 Hora de verano romance
gpg: usando EDDSA clave 18DE635E4EAE6E68DFAD2F7CD730BE196E466101
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: 18DE 635E 4EAE 6E68 DFAD 2F7C D730 BE19 6E46 6101
```

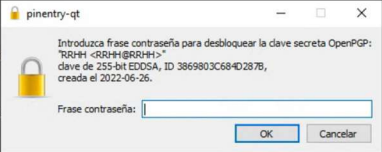
Así mismo, se requiere firmar el siguiente mensaje con la clave correspondiente de las anteriores, simulando que eres personal de RRHH.

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

```
C:\Users\dmore\Desktop\Ciberseguridad\Criptografia\Práctica\Ejercicio 10>gpg --verify MensajeRespoDeRaulARRHH.sig
gpg: Firmado el 06/26/22 13:47:01 Hora de verano romance
gpg: usando EDDSA clave 18DE635E4EAE6E68DFAD2F7CD730BE196E466101
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: 18DE 635E 4EAE 6E68 DFAD 2F7C D730 BE19 6E46 6101

C:\Users\dmore\Desktop\Ciberseguridad\Criptografia\Práctica\Ejercicio 10>gpg --output mensaje
gpg: no se puede abrir 'mensaje_a_firmar.doc': No such file or directory
gpg: mensaje_a_firmar.doc: clear-sign failed: No such file or directory

C:\Users\dmore\Desktop\Ciberseguridad\Criptografia\Práctica\Ejercicio 10>gpg --output mensaje
El fichero 'mensaje_a_firmar.sig' ya existe. ¿Sobreescribir? (s/N) s
```



The image shows a terminal window with GPG commands and their output. The first command is a verification of a signature, which succeeds. The second command is an attempt to sign a file named 'mensaje\_a\_firmar.doc', which fails because the file does not exist. The third command is another attempt to sign a file, which also fails for the same reason. The fourth command is a prompt to overwrite an existing signature file. Overlaid on the terminal is a 'pinentry-qt' dialog box asking for a password to unlock a secret key. The dialog shows the key ID '3869803C684D287B' and the creation date '2022-06-26'.

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.

Buscamos las ID de cada clave:

```
C:\Users\dmore\Desktop\Ciberseguridad\Criptografia\Práctica\Ejercicio 10>gpg --list-keys
[keyboard]
-----
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    [ absoluta ] RRHH <RRHH@RRHH>
sub    cv25519 2022-06-26 [E]
```

Y ciframos:

```
C:\Users\dmore\Desktop\Ciberseguridad\Criptografia\Práctica\Ejercicio 10>gpg --output mensaje_a_cifrar3.txt.gpg --encrypt --recipient 1BDE635E4EAE6E68DFAD2F7CD730BE196E466101 --recipient F2B1D0E8958DF2D3BDB6A1053869803C684D287B mensaje_a_cifrar3.txt
gpg: 25D6D02940358650: No hay seguridad de que esta clave pertenezca realmente al usuario que se nombra
sub   cv25519/25D6D02940358650 2022-06-26 Pedro Pedrito Pedro <pedro.pedrito.pedro@empresa.com>
Huella clave primaria: 1BDE 635E 4EAE 6E68 DFAD 2F7C D730 BE19 6E46 6101
Huella de subclave: 8E8C 6669 AC44 3271 42BC C244 25D6 D029 4035 B650

No es seguro que la clave pertenezca a la persona que se nombra en el
identificador de usuario. Si *realmente* sabe lo que está haciendo,
puede contestar sí a la siguiente pregunta.

¿Usar esta clave de todas formas? (s/N) s
C:\Users\dmore\Desktop\Ciberseguridad\Criptografia\Práctica\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
793eb00cc76d10fc00475eb76bfbcb1273303882609957c4c0ae2c4f5ba670a4126f2f14
a9f4b6f41aa2edba01b4bd586624659fca82f5b4970186502de8624071be78cccf573d
896b8eac86f5d43ca7b10b59be4acf8f8e0498a455da04f67d3f98b4cd907f27639f4b1
df3c50e05d5bf63768088226e2a9177485c54f72407fdf358fe64479677d8296ad38c6f
177ea7cb74927651cf24b01dee27895d4f05fb5c161957845cd1b5848ed64ed3b0372
2b21a526a6e447cb8ee
```



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:** [Public RSA key at 0x22918C8EA50](#)

**Cifrado:**

[5cbde252eadf5d607115711d9a427b3f80c5e58c4d3460188d8f9400fe1b2f9af6dabc6fec566281e7064b13d28ba8ff22859bf62b457d6e7db828d1637f7367cad9e71d9793997f9d7b16ef1c46f01ba47c39c30758fa27bda873dfdd8872c23f99aba2306c5300dead4c9cc145f82e6dbeb8aec64ab66cacd5d3229516557d50afecad02ab36cbd2dd9201001c4169fc99718a2b79e1d655c4d714fe7b4f07e01d9d3a2786990d0bf6ee7d43987f328a3ba81b1953968ee2721bec331d5893e63bdadf1cf0e67197d2cca08df21a71ebd98c35603f3cd9eaca09d397d2ddeb14ea88a75d06b7085d7267d3af38029fa8fb6958c346f1f7182fe10a4d0c7cd1](#)

**El padding OAEP incluye una combinación de hash y un vector de inicialización (IV) aleatorio, lo que asegura que los cifrados sean únicos incluso si el texto cifrado es el mismo.**

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/f5nJhAt2S](#)

¿Qué estamos haciendo mal?

**El nonce (o IV) debe ser único para cada mensaje cifrado. Si se reutiliza el mismo nonce con la misma clave para cifrar diferentes mensajes, se compromete la seguridad del sistema, permitiendo potencialmente ataques como el de "nonce reuse".**

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 hex:**

[5dcbb6261d0fba29ce39431e9a013b34cbca2a4e04bb2d90149d61f4afd04d65e2abdd9d84bba6eb8307095f5078fbfc16256d](#)

**Texto cifrado b64:**

[Xcu2Jh0PuinOOUMemgE7NMvKKk4Euy2QFJ1h9K/QTWxiq92dhLum64MHCv9QePv8FiVt](#)

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

**Valor de la firma en hexadecimal:**

a4606c518e0e2b443255e3626f3f23b77b9d5e1e4d6b3dcf90f7e118d6063950a23885c6de  
ce92aa3d6eff2a72886b2552be969e11a4b7441bdeadc596c1b94e67a8f941ea998ef08b2c  
b3a925c959bcaae2ca9e6e60f95b989c709b9a0b90a0c69d9eaccd863bc924e70450ebbbb8  
7369d721a9ec798fe66308e045417d0a56b86d84b305c555a0e766190d1ad0934a1befbbe  
031853277569f8383846d971d0daf05d023545d274f1bdd4b00e8954ba39dacc4a0875208f  
36d3c9207af096ea0f0d3baa752b48545a5d79cce0c2ebb6ff601d92978a33c1a8a707c1ae1  
470a09663acb6b9519391b61891bf5e06699aa0a0dbae21f0aaaa6f9b9d59f41928d

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?

**Clave AES(hex):**

e716754c67614c53bd9bab176022c952a08e56f07744d6c9edb8c934f52e448a

**Clave AES(b64):** 5xZ1TGdhTFO9m6sXYCLJUqCOVvB3RnbJ7bjNPUuRlo=

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? ¿Para qué algoritmo se ha definido la clave?

**Algoritmo: A**

Si A se refiere a AES, este campo está diciendo que la clave embebida es una clave AES. Este algoritmo puede tener variantes como AES-128, AES-192, o AES-256, dependiendo de la longitud de la clave.

¿Para qué modo de uso se ha generado?

**Modo de uso: B**

**B podría indicar que la clave se puede usar tanto para cifrar como para descifrar datos. Esto es importante para aplicaciones que necesitan realizar ambas operaciones con la misma clave.**

¿Es exportable?

**Exportabilidad: S**

**S puede estar diciendo que la clave puede ser exportada de su entorno actual de manera segura, lo que es útil si la clave necesita ser transferida a otro sistema o dispositivo de manera segura.**

¿Para qué se puede usar la clave?

**Uso de la clave: D0**

**Por ejemplo, D0 podría estar indicando que la clave es para "Data Encryption", es decir, se usará para cifrar y descifrar datos.**

¿Qué valor tiene la clave?

**La clave es: c1c1c1c1c1c1c1c1c1c1c1c1c1c1c1c1**