Práctica de criptografía

Ejercicio 1

Para la realización de este ejercicio se ha utilizado el código perteneciente al fichero: **ejercicio1.py**

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?

Clave del fichero properties:
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?

Clave final en memoria:
8653f75d31455c0

Ejercicio 2

Para la realización de este ejercicio se ha utilizado el código perteneciente al fichero: **ejercicio2.py**

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?

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? ¿Cuánto padding se ha añadido en el cifrado?

Se añade 01 de padding.

Ejercicio 3

Para la realización de este ejercicio se ha utilizado el código perteneciente al fichero: **ejercicio3.py**

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.

Mensaje cifrado en HEX
69ac4ee7c4c552537a00a19bcaf7f0aaed7c9c8f769956a09bce6fadef6c3535f2211c9467067cf5c4a842ab

Mensaje cifrado en B64
aaxO58TFUlN6AKGbyvfwqu18nI92mVagm85vre9sNTXyIRyUZwZ89cSoQqs=

¿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 la realización del ejercicio, se ha utilizado el código perteneciente al archivo: **ejercicio3-1.py**

Para mejorar este ejercicio, generamos un nonce de manera aleatoria además de unos datos asociados que nos permiten garantizar la confidencialidad.

Ejercicio 4

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

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmlvIjoiRG9uIFBlcGl0byBkZSBsb3MgcGFsb3RlcyIsInJvbCI6ImlzTm9ybWFsIiwiaWF0IjoxNjY3OTMzNTMzfQ.gfhw0dDxp6oixMLXXRP97W4TDTrv0y7B5YjD0U8ixrE

¿Qué algoritmo de firma hemos realizado?
Se ha utilizado un algoritmo HS256

¿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.eyJ1c3VhcmlvIjoiRG9uIFBlcGl0byBkZSBsb3MgcGFsb3RlcyIsInJvbCI6ImlzQWRtaW4iLCJpYXQiOjE2Njc5MzM1MzN9.krgBkzCBQ5WZ8JnZHuRvmnAZdg4ZMeRNv2CIAODlHRI

¿Qué está intentando realizar?

Está tratando de escalar los privilegios del usuario Don Pepito de los palotes a Admin

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

¿Qué ocurre si intentamos validarlo con pyjwt?

Que no se valida ya que la firma es inválida

Ejercicio 5

Para la realización de este ejercicio se ha utilizado el código perteneciente al fichero: **ejercicio5.py**

El siguiente hash se corresponde con un SHA3 del texto “En KeepCoding aprendemos cómo protegernos con criptografía”.

bced1be95fbd85d2ffcce9c85434d79aa26f24ce82fbd4439517ea3f072d56fe

¿Qué tipo de SHA3 hemos generado?

SHA3-256

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

4cec5a9f85dcc5c4c6ccb603d124cf1cdc6dfe836459551a1044f4f2908aa5d63739506f6468833d77c07cfd69c488823b8d858283f1d05877120e8c5351c833

¿Qué hash hemos realizado?

SHA2-512

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?

302be507113222694d8c63f9813727a85fef61a152176ca90edf1cfb952b19bf

La propiedad de difusión, ya que únicamente añadimos un punto al texto pero el hash cambia por completo.

Ejercicio 6

Para la realización de este ejercicio se ha utilizado el código perteneciente al fichero: **ejercicio6.py**

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.

857d5ab916789620f35bcfe6a1a5f4ce98200180cc8549e6ec83f408e8ca0550

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?

SHA-1 se considera inseguro desde 2005 debido a varias debilidades criptográficas, por lo que nunca deberíamos usarlo.

Los ataques de colisión son más viables con SHA-1. En 2017, investigadores demostraron una colisión práctica en SHA-1.

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?

Podemos fortalecerlo a través de métodos como el salting o el peppering, que agregan valores a las contraseñas antes de hacer el hash para fortalecer la seguridad

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

Para mejorar aún más la seguridad del almacenamiento de contraseñas, se pueden usar algoritmos de hashing diseñados específicamente para este propósito. Como primera opción utilizaremos el Argon2, aunque existen otras opciones como el bcrypt o el scrypt

Ejercicio 8

Tenemos la siguiente API REST, muy simple.

[Código de la API]

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?

Para asegurar la confidencialidad de los datos sensibles, como el número de tarjeta, podemos utilizar AES/GCM, que proporciona confidencialidad y autenticación de los datos cifrados.

Además, para garantizar que los mensajes no han sido modificados, podemos utilizar HMAC.

Ejercicio 9

Para la realización de este ejercicio se ha utilizado el código perteneciente al fichero: **ejercicio9.py**

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: 5244db
KCV SHA256: db7df2

Ejercicio 10

El responsable de Raúl, Pedro, ha enviado este mensaje a RRHH:

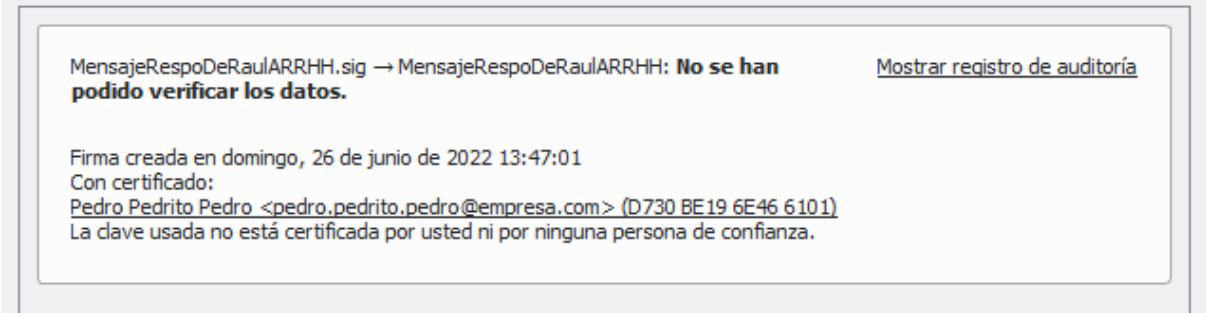
Se debe ascender inmediatamente a Raúl. Es necesario mejorarle sus condiciones económicas un 20% para que se quede con nosotros.

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.

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.

Para la realización de este ejercicio se ha utilizado el código perteneciente al fichero: **NuevoMensajeFirmado.txt.sig**

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

Para la realización de este ejercicio se ha utilizado el código perteneciente al fichero: **MensajeFinalCifrado.txt.sig**

Ejercicio 11

Para la realización de este ejercicio se ha utilizado el código perteneciente al fichero: **ejercicio11.py**

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
df3c50e05d5bf63768088226e2a9177485c54f72407fdf358fe64479677d8296ad38c6f177ea7
cb74927651cf24b01dee27895d4f05fb5c161957845cd1b5848ed64ed3b0372
2b21a526a6e447cb8ee

Al ejecutar el código con el texto cifrado del ejercicio me daba este error;
ValueError: Incorrect decryption.

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?
Porque el OAEP utiliza valores aleatorios para el padding en cada operación de cifrado

Ejercicio 12

Para la realización de este ejercicio se ha utilizado el código perteneciente al fichero: **ejercicio12.py**

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 debe ser único y no repetirse nunca para cada comunicación encriptada con la misma clave.

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.

Tag:
6120e37aa4c3ecfd9261640dcc46410d

CipherText_Hex:
5dcbb6261d0fba29ce39431e9a013b34cbca2a4e04bb2d90149d61f4afd04d65e2abdd9d84bba6eb8307095f5078fbfc16256d

CipherText_B64:
Xcu2Jh0PuinOOUMemgE7NMvKKk4Euy2QFJ1h9K/QTWXiq92dhLum64MHCV9QePv8FiVt

Ejercicio 13

Para la realización de este ejercicio se ha utilizado el código perteneciente al fichero: **ejercicio13.py**

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:
b'a4606c518e0e2b443255e3626f3f23b77b9d5e1e4d6b3dcf90f7e118d6063950a23885c6dec
e92aa3d6eff2a72886b2552be969e11a4b7441bdeadc596c1b94e67a8f941ea998ef08b2cb3a
925c959bcaae2ca9e6e60f95b989c709b9a0b90a0c69d9eaccd863bc924e70450ebbbb87369
d721a9ec798fe66308e045417d0a56b86d84b305c555a0e766190d1ad0934a1befbbe031853
277569f8383846d971d0daf05d023545d274f1bdd4b00e8954ba39dacc4a0875208f36d3c920
7af096ea0f0d3baa752b48545a5d79cce0c2ebb6ff601d92978a33c1a8a707c1ae1470a09663
acb6b9519391b61891bf5e06699aa0a0dbae21f0aaaa6f9b9d59f41928d'

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

Para la realización de este ejercicio se ha utilizado el código perteneciente al fichero: **ejercicio13b.py**

Ejercicio 14

Para la realización de este ejercicio se ha utilizado el código perteneciente al fichero: **ejercicio14.py**

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: e716754c67614c53bd9bab176022c952a08e56f07744d6c9edb8c934f52e448a

Ejercicio 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?

- AES

¿Para qué algoritmo se ha definido la clave?

- AES

¿Para qué modo de uso se ha generado?

- D0 - Symmetric Key for Data Encryption

¿Es exportable?

- Sí

¿Para qué se puede usar la clave?

- Both Encrypt & Decrypt / Wrap & Unwrap

¿Qué valor tiene la clave?

- c1c1c1c1c1c1c1c1c1c1c1c1c1c1c1c1