PRACTICA DE CRIPTOGRAFIA

**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. 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?**

Una de las primeras cosas que aprendimos es que los XOR tienen propiedades conmutativas. Sabiendo esto el ejercicio se vuelve bastante sencillo. Tenemos dos valores, el de la clave fija en codigo, y el valor de la clave final (en memoria).

Para sacar el valor de la clave restante solo tenemos que hacer un XOR entre la clave fija y la clave final. El valor que esta nos devuelve es 20553975C31055ED.

Este seria el valor que el key manager puso en properties.

Si hicieramos un XOR entre la clave fija y el resultado que obtuvimos de la clave final, veriamos que el resultado seria la clave final que ya nos habian dado en un principio.

La segunda pregunta es basicamente lo mismo es solamente hacer el XOR entre las dos claves dadas y asi obtendremos en un valor hexadecimal el resultado de la clave en memoria.

08653f75d31455c0

**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/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? ¿Cuánto padding se ha añadido en el cifrado?**

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

Si lo desciframos obtenemos esto:

El texto en claro es: Esto es un cifrado en bloque típico. Recuerda, vas por el buen camino. Ánimo.

Si decidimos cambiar el padding en el cifrado y sacamos la funcion de unpad, podemos ver cuanto padding se le esta rellenando al bloque. Con el padding x923 se le rellena lo siguiente:

0100000000000000000000000000000010

Si quisieramos ver el padding del pkcs7 seria el siguiente: 0110101010101010101010101010101010

**Ejercicio3**

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?**

Para mejorar el sistema, lo unico que debemos hacer es no tener el nonce fijo. El nonce no se debe repetir nunca por lo cual es mucho mejor generar el nonce de forma aleatoria. Tambien existe una version mejor a la del chacha 20, el chacha 20 Poly, que es muchisimo mejor que la anterior ya que mediante su uso de tag tenemos mucha mas seguridad, es decir garantizamos su confidencialidad y integridad.

**Ejercicio 4**

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

**eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmlvIjoiRG9uIFBlcGl0byBkZSB sb3MgcGFsb3RlcyIsInJvbCI6ImlzTm9ybWFsIiwiaWF0IjoxNjY3OTMzNTMzfQ.gfhw0 dDxp6oixMLXXRP97W4TDTrv0y7B5YjD0U8ixrE**

**¿Qué algoritmo de firma hemos realizado?**

El algoritmo de firma que tiene este JWT es un HMAC 256

**¿Cuál es el body del jwt?**

El body del JWT es el siguiente:

{

"usuario": "Don Pepito de los palotes",

"rol": "isNormal",

"iat": 1667933533

}

Un hacker está enviando a nuestro sistema el siguiente jwt:

**eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmlvIjoiRG9uIFBlcGl0byBk ZSBsb3MgcGFsb3RlcyIsInJvbCI6ImlzQWRtaW4iLCJpYXQiOjE2Njc5MzM1MzN9 .krgBkzCBQ5WZ8JnZHuRvmnAZdg4ZMeRNv2CIAODlHRI**

**¿Qué está intentando realizar?**

El hacker esta intentando de hacer un cambio de rol, escalar privilegios.

**¿Qué ocurre si intentamos validarlo con pyjwt?**

No se puede validar, porque la firma es inválida.

**Ejercicio 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?**

Generamos un sha3 256, es facil de saberlo por la longitud del mismo.

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

**4cec5a9f85dcc5c4c6ccb603d124cf1cdc6dfe836459551a1044f4f2908aa5d63739506f 6468833d77c07cfd69c488823b8d858283f1d05877120e8c5351c833**

**¿Qué hash hemos realizado?**

Realizamos un SHA2 512, porque la longitud es de 128 caracteres, que equivalen a 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?

El hash es el siguiente:

302be507113222694d8c63f9813727a85fef61a152176ca90edf1cfb952b19bf

Calquier minimo cambio en un bit produce un hash totalmente distinto. Aca el “.” es el caracter diferente a la frase del principio y esto hace que los hashes sean totalmente distintos.

**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.

El hmac-256 del texto: Siempre existe más de una forma de hacerlo, y más de una solución válida. es el siguiente:

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?**

Es una mala opcion porque es un tipo de algoritmo vulnerable, hoy en dia los que se recomiendan usar son: Sha2 o Sha3.

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?**

Una buena manera de fortalecer los hash es haciendo uso de Salt y Pepper, son un valor agregado que le metemos al hash el cual es aleatorio.

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

**¿Qué propondrías?**

La mejor manera de mejorar o fortalecer el sistema es utilizando un mac. es exactamente lo mismo que un tag, su funcion es la de brindar un codigo de autenticacion, con este podemos garantizar la confidencialidad autenticacion e integridad de los datos.

**Ejercicio 8**

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?**

La mejor manera de conseguir todas las cosas mencionadas es con algoritmos que nos brinden un tag o mac. Con estos podemos garantizar la confidencialidad e integridad de los mensajes. Un AES/GCM para criptografia simetrica en bloque o un CHACHA 20 Poly1305 para criptografia en flujo.

**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.

El KCV de la clave es el siguiente:

KCV: 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.**

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.**

**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 96b4a8a20d5dbc639e9d83a53681e6d96f76a0e4c279f0dffa76a329d04e3d3d4ad629793eb00cc76d10fc00475eb76bfbc1273303882609957c4c0ae2c4f5ba670a4126f2f14a9f4b6f41aa2edba01b4bd586624659fca82f5b4970186502de8624071be78ccef573d896b8eac86f5d43ca7b10b59be4acf8f8e0498a455da04f67d3f98b4cd907f27639f4b1df3c50e05d5bf63768088226e2a9177485c54f72407fdf358fe64479677d8296ad38c6f177ea7cb74927651cf24b01dee27895d4f05fb5c161957845cd1b5848ed64ed3b0372 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?**

El resultado es:

**“e2cff885901a5449e9c448ba5b948a8c4ee377152b3f1acfa0148fb3a426db72”**

cuando volvemos a cifrar el texto es distinto porque RSA utiliza esta caracteristica como seguridad.

**Ejercicio 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?**

Estamos teniendo un nonce fijo, esto es algo que no se puede hacer nunca, no se debe de repetir jamas el nonce, tiene que ser aleatorio.

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.

El texto en claro es:

**He descubierto el error y no volveré a hacerlo mal**

Texto cifrado en base64:

Xcu2Jh0PuinOOUMemgE7NMvKKk4Euy2QFJ1h9K/QTWXiq92dhLum64MHCV9QePv8FiVt

Texto cifrado en hexadecimal: 5dcbb6261d0fba29ce39431e9a013b34cbca2a4e04bb2d90149d61f4afd04d65e2abdd9d84bba6eb8307095f5078fbfc16256d

**Ejercicio 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.

**Las firmas en hexadecimal son:**

ed25519:

* 626633323539326463323335613236653331653233313036336131393834626237356666643964633535353063663330313035393131636134353630646162353261626234306534663765326433616638323861626163313436376439
* 35643636386138303339356530613731633531373938626435343436396237
* 33363064
* PKCS: 

**Ejercicio 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?**

Se obtuvo esta clave:

e716754c67614c53bd9bab176022c952a08e56f07744d6c9edb8c934f52e448a

**Ejercicio 15.**

Nos envían un bloque TR31:

**D0144D0AB00S000042766B9265B2DF93AE6E29B58135B77A2F616C8D515ACDBE6A5626F79FA7B4071E9EE1423C6D7970FA2B965D18B23922B5B2E5657495**

**03CD857FD37018E111B**

Donde la clave de transporte para desenvolver (unwrap) el bloque es: **A1A10101010101010101010101010102**

**¿Con qué algoritmo se ha protegido el bloque de clave?**

El bloque de clave se protegio con un algoritmo de tipo AES

**¿Para qué algoritmo se ha definido la clave?**

La clave se definio para un algoritmo de tipo AES

**¿Para qué modo de uso se ha generado?**

Se genero la clave para cifrar y descifrar

**¿Es exportable?**

Si pero es sensible, exportable solo bajo clave no confiable

**¿Para qué se puede usar la clave?**

Es una clave simetrica que se puede usar para cifrar y descifrar datos.

**¿Qué valor tiene la clave?**

La clave tiene este valor: c1c1c1c1c1c1c1c1c1c1c1c1c1c1c1c1